Optimizations for the Query Language SPARQL-LD

Thanos Yannakis

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, Heraklion, GR-70013, Greece

Thesis Advisor: Associate Prof. *Yannis Tzitzikas*

Dr. *Pavlos Fafalios*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

The work has been supported by the Foundation for Research and Technology - Hellas (FORTH), Institute of Computer Science (ICS).

University of Crete
Computer Science Department

Optimizations for the Query Language SPARQL-LD

Thesis submitted by
Thanos Yannakis
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: Thanos Yannakis

Committee approvals: Yannis Tzitzikas
Associate Professor, University of Crete
Thesis Supervisor

Dimitris Plexousakis
Professor, University of Crete
Committee Member

Giorgos Flouris
Affiliated Researcher, FORTH-ICS
Committee Member

Departmental approval: Antonis Argyros
Professor, University of Crete
Director of Graduate Studies

Heraklion, 2017

Optimizations for the Query Language SPARQL-LD

Abstract

Linked Data is a method for publishing structured data in the Web for assisting their linking and integration. A constantly increasing number of organizations and owners publish their data on the Web as Linked Data and SPARQL is the standard query language. However the majority of SPARQL implementations require the data to be available in advance in main memory or accessible through a SPARQL endpoint.

`SPARQL-LD` is an extension of SPARQL 1.1 (designed by FORTH-ICS) that overcomes this restriction and allows fetching and querying RDF data from any Web source and format i.e., RDFa, JSON-LD, Microdata and Microformats. Using `SPARQL-LD` one can query a dataset corresponding to an RDF dump, or a dataset corresponding to the partial results of a query (i.e., discovered at query execution time), or RDF data that are dynamically created by Web Services. This functionality can motivate content owners to adopt the Linked Data principles and enrich their digital content and services with RDF, for having their data directly queryable via SPARQL–LD without having to create and maintain an operational SPARQL endpoint.

In this thesis we focus on optimizations for `SPARQL-LD` in particular on: (a) methods for exploring the syntactic variations of graph patterns in a SPARQL query in order to choose a near to optimal execution plan without the use of statistics (to this end we utilize query reordering techniques, using selectivity estimation procedures on new unbound variables for increasing efficiency and decreasing intermediate results and thus the number of calls to remote sources), and (b) methods for parallelizing RDF data retrieval by using `SPARQL-LD` for efficient data caching.

Finally, we report experimental results on real datasets for evaluating the efficiency as well as the quality of the proposed optimizations. The results showed improved efficiency on SPARQL queries in comparison to existing methods.

Βελτιστοποιήσεις για τη Γλώσσα Επερώτησης Διασυνδεδεμένων Δεδομένων SPARQL-LD

Περίληψη

Τα Διασυνδεδεμένα Δεδομένα (Linked Data) είναι ένας τρόπος δημοσίευσης δεδομένων στο διαδίκτυο που διευκολύνει τη διασύνδεση (μέσω της χρήσης URIs αντί απλών τιμών) και την ολοκλήρωσή τους. Υπάρχει ένας διαρκώς αυξανόμενος αριθμός οργανισμών ή επιχειρήσεων που δημοσιεύουν τα δεδομένα τους ως διασυνδεδεμένα δεδομένα (Linked Data) τα οποία είναι επερωτήσιμα μέσω της SPARQL που είναι η στάνταρ γλώσσα επερωτήσεων του Σημασιολογικού Ιστού (Semantic Web). Ωστόσο, η χρήση της SPARQL προϋποθέτει ότι από τα δεδομένα προς επερώτηση είναι διαθέσιμα εκ των προτέρων στην κύρια μνήμη ή σε κάποιο σημείο σύνδεσης SPARQL (SPARQL endpoint).

Για να δώσουμε μεγαλύτερη ευελιξία, σε αυτήν την εργασία επικεντρωνόμαστε στην SPARQL-LD, μια επέκταση της SPARQL 1.1 που σχεδιάστηκε στο Ινστιτούτο Πληροφορικής, η οποία προσφέρει τη δυνατότητα επερώτησης σε απομακρυσμένα δεδομένα ακόμα και αν δεν φιλοξενούνται από ένα σημείο σύνδεσης SPARQL. Χρησιμοποιώντας τη SPARQL-LD μπορεί κανείς να επερωτήσει σύνολα δεδομένων (datasets) που είναι προσβάσιμα ως RDF dumps, ως ενσωματωμένα δεδομένα σε ιστοσελίδες, δηλαδή σε μορφή RDFa, JSON-LD, Microdata ή Microformat, καθώς και σε δεδομένα που αντιστοιχούν σε μερικά αποτελέσματα μιας επερώτησης (δηλαδή, που ανακτήθηκαν κατά το χρόνο εκτέλεσης της επερώτησης), ή που δημιουργήθηκαν δυναμικά από υπηρεσίες διαδικτύου. Αυτή η λειτουργικότητα μπορεί να δώσει κίνητρο στους ιδιοκτήτες περιεχομένου να υιοθετήσουν τις αρχές των διασυνδεδεμένων δεδομένων και να εμπλουτίσουν το ψηφιακό τους περιεχόμενο και υπηρεσίες τους με RDF, αφού έτσι τα δεδομένα τους θα είναι άμεσα επερωτήσιμα μέσω της SPARQL-LD χωρίς να χρειάζονται να δημιουργήσουν και να συντηρούν ένα λειτουργικό σημείο σύνδεσης SPARQL.

Εν συνεχεία, επικεντρωνόμαστε σε βελτιστοποιήσεις που αφορούν τη SPARQL-LD, συγκεκριμένα σε: (α) μεθόδους που διερευνούν συντακτικές παραλλαγές των μοτίβων γράφων (Graph Patterns) μιας επερώτησης SPARQL για την επιλογή του σχεδόν βέλτιστου πλάνου εκτέλεσης χωρίς τη χρήση στατιστικών στοιχείων (για το σκοπό αυτό εφαρμόζουμε τεχνικές αναδιάταξης επερωτήσεων βασισμένων σε τεχνικές εκτίμησης της εκλεκτικότητας νέων - μη δεσμευμένων - μεταβλητών με στόχο την επιτάχυνση καθώς και τη μείωση των απομακρυσμένων κλήσεων), και (β) τεχνικές παράλληλης ανάκτησης RDF δεδομένων από το διαδίκτυο με την χρήση απομακρυσμένων επερωτήσεων της SPARQL-LD για αποδοτικότερη προσωρινή αποθήκευση των δεδομένων.

Τέλος, αναφέρουμε πειραματικές μετρήσεις που πραγματοποιήθηκαν πάνω σε πραγματικά σύνολα δεδομένων για την αξιολόγηση της απόδοσης καθώς και της ποιότητας των προτεινόμενων βελτιστοποιήσεων, όπου παρατηρήθηκε σημαντική βελτίωση της επίδοσης κατά την χρήση απομακρυσμένων κλήσεων σε σχέση με άλλες μεθόδους.

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω θερμά τον επόπτη καθηγητή μου κ. Γιάννη Τζίτζικα για την άψογη συνεργασία και ουσιαστική συμβολή του στην ολοκλήρωση της παρούσας μεταπτυχιακής εργασίας. Ακόμη θέλω να εκφράσω τις ευχαριστίες μου στον κ. Δημήτρη Πλεξουσάκη και στον κ. Γιωργο Φλουρή για την προθυμία τους να συμμετέχουν στην τριμελή επιτροπή. Επίσης θα ήθελα να ευχαριστήσω ιδιαίτερα τον μεταδιδακτορικό ερευνητή Παύλο Φαφαλιό για την προθυμία και την υπομονή καθ΄ όλη την διάρκεια της παρούσας εργασίας.

Ακόμα ευχαριστώ το Ινστιτούτο Πληροφορικής του Ιδρύματος Τεχνολογίας και Έρευνας για την πολύτιμη υποστήριξη σε υλικοτεχνική υποδομή και τεχνογνωσία, καθώς και για την υποτροφία που μου προσέφερε κατα τη διάρκεια της μεταπτυχιακής μου εργασίας.

Στο σημείο αυτό θα ήθελα να ευχαριστήσω τους γονείς μου για την συμπαράσταση και την υποστήριξη που μου έδωσαν όλα αυτά τα χρόνια. Επειτα θα ήθελα να ευχαριστήσω τους συναργάτες και φίλους που ήταν δίπλα μου και πάντα πρόθυμοι να με βοηθήσουν καθ΄ όλη την διάρκεια των σπουδών μου.

…

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Topic and Motivation

The content of the World Wide Web (WWW) is currently formatted in a natural language, mainly through HTML. Even though such a language is human-readable and human-understandable, the machines or else the software agents are only able to read this information. The machine-intelligibility cannot be achieved with the current technology.

This gap is called to be solved through the Semantic Web, an extension of the WWW. The term was coined by the Tim Berners-Lee, the inventor of the WWW and the director of the Word Wide Web Consortium (W3C), and aims at converting unstructured and semi-structured documents into semantically structured knowledge, that can be processed directly and indirectly by machines. The promoted formats give the ability to the machines to interpret the content of the web page and find, share and integrate information more easily.

Linked Data is about using the Web to connect related data that wasn't previously linked, or using the Web to lower the barriers to linking data currently linked using other methods. More specifically, Wikipedia defines Linked Data as "a term used to describe a recommended best practice for exposing, sharing, and connecting pieces of data, information, and knowledge on the Semantic Web using URIs and RDF". The goal is to extend the Web with a data commons by publishing various open datasets as RDF on the Web and by setting RDF links between data items from different data sources [18].

Linked Data and Web technologies have essentially transformed the Web from a publishing-only environment into a vibrant place for information dissemination where data is exchanged, integrated, and materialized in distributed repositories. In this emerging global data space, the real benefit we can get from the available semantic data relies on our capability to access and interpret them. Most of the applications require data integration as well as reconstruction in order to adapt them to other contexts of use than the ones they were originally envisioned for. There are a variety of techniques to manipulate and use such data based on their meaning. A central issue in this context is the meaningful querying and storage of semantic data. With this in mind we are striving for efficient query and storage techniques for linked data expressed in RDF, the base component of linked open data and therefore the standard data model for the Web of Data.

However, the current state of the available commercial RDF stores known as SPARQL endpoints, shows problematic query performance and responsiveness, typically caused by bad query plans, especially in complex queries. Since SPARQL language is declarative, a SPARQL query can be written differently with several orders of expressions. However, the execution time for the different orders that are generated from the same SPARQL statement can vastly vary in efficiency. Sometimes, simple re-orderings can reduce the querying time considerably. Therefore Query Planning, which evaluates the possible query plans and finds a best one for the query engine, is regarded as an essential task in query optimizer.

An important question has risen, how one can efficiently access and query this constantly increasing body of knowledge? SPARQL [8] is a standard query language (W3C Recommendation) for retrieving and manipulating RDF data from the Web of Data. However, the majority of SPARQL implementations require the data to be available in advance, i.e., to exist in main memory, or in a RDF repository accessible through a SPARQL endpoint. Nonetheless, Linked Data exists in the Web in various forms; even an HTML Web page can contain RDF data through RDFa [7], or JSON-LD [1], or Microdata [5], or Microformats [6] or RDF data that may be dynamically created by Web Services.

Our work is based on `SPARQL-LD` [10], a generalization of SPARQL 1.1 Federated Query [9] that allows to *directly* and *flexibly* exploit this wealth of data. `SPARQL-LD` extends the applicability of the `SERVICE` operator (of SPARQL 1.1 Federated Query) enabling to query any HTTP Web source containing RDF data. `SPARQL-LD` does not require the named graphs to have been declared, thus one can even fetch and query a dataset returned by a portion of the query (i.e., whose URI is derived at query execution time).

1.2 Objectives and Approach

In this thesis we focus on query optimizations for `SPARQL-LD`. There are two main approaches for query optimization, i.e., on how to evaluate the cost of SPARQL expressions. The most common approaches use pre-computed statistics [45] and heuristics [59]. The first approach calculates certain summary data on RDF resources, usually using histogram based methods, and then utilize the data to evaluate the cost of query plans. Heuristic-based methods first define certain heuristics according to the observation on RDF data sources and then apply these heuristics to estimate the cost. Each approach has some advantages and also some limitations. Statistics-based methods are usually more expensive in terms of implementation, required resources (time, storage) and maintenance, particularly for large scale evolving semantic data over the Web, in spite of that, such methods preserve relatively higher accuracy. Heuristic-based approaches are easier to implement and cost less, but can be less efficient.

Nevertheless, in many use cases, where SPARQL users are accessing LOD data sources, typically reachable over an URI and often freshly loaded, the database may not have the needed statistics (e.g., histograms) for cost-based optimization. Moreover, in RDF it is not immediately clear on what to create statistics, as the data is essentially a directed labelled graph, where the same predicates may be used between multiple sub-classes of subjects/objects (and where these sub-classes are not explicitly declared or recognizable), and in which predicates themselves may also re-appear as subjects and objects, mixing data and metadata in this one big graph. With this in mind the job of a SPARQL query optimizer is significantly complex, not only because of the usually large amount of self joins, but also because it is not trivial to estimate the join hit-ratios in the SPARQL case. As a result, SPARQL endpoints, even if they rely on cost-based statistics for certain kinds of predicates (such as selections, or certain well-known joins) will in many other cases have to rely on heuristics anyway.

In the case of `SPARQL-LD` we could rely on statistics to improve the efficiency of the system over specific resources, e.g., the statistics that were gathered from wikipedia, yago and wikidata in section 5.1. However due to the dynamic nature of `SPARQL-LD` to be able to query any HTTP resource and endpoints, we will use a heuristics approach that applies general rules and filters to enable querying over different datasets.

Thus, in this thesis we present a heuristic approach to solve this problem by devising heuristics based query optimization techniques without the need of any knowledge of the stored dataset. Our work focuses on static Group Graph Pattern (GGP) optimization for SPARQL queries and main memory graph implementations of RDF data. Our presented optimization approaches are inspired from join re-ordering strategies using selectivity estimation [58]. To this end, we exploit the syntactic and structural variations of the triple and `SERVICE` patterns and operators in a SPARQL query in order to choose a near to optimal execution plan without the need of any statistics. Based solely on the syntax of a SPARQL query, we can decide which `SERVICE` calls to evaluate first (by selecting the `SERVICE` call with the lowest selectivity) in order to reduce the intermediate results, hence increasing query performance.

The main problem we are going to tackle in this thesis is best explained by a simple example. Consider the query displayed in Figure 1.1 which represents a series of `SERVICE` calls executed over RDF data to retrieve information about scientific journals. In more detail the first federated query is executed against the endpoint of dbpedia[1] and searches for academic journals that focus on the subject of toxicology and retrieves their labels as well as *same-as* connections to other endpoints. Similarly, the second federated query is executed against the

[1]`https://dbpedia.org/sparql`

endpoint of wikidata[2] and searches for journals that are published in 1978 and originate from United Kingdom. While the third federated query uses a variable operator to fetch all the relative information from the dereferencable URIs of the journals recovered in the previous `SERVICE` calls. The question is in which order a query engine should execute the three `SERVICE` calls. Given the research on join order strategies that has been pursued for relational database systems, we can safely state that a query engine should execute first the second triple pattern as its result set is considerably smaller compared to the result set of the first triple pattern. We can observe that the second federated query is more **restrictive** than the first one. With this in mind the third federated query needs to preserve its order in order to be correctly evaluated since it makes use of a variable `SERVICE` operator. Therefore, a static optimizer is in need to reverse the `SERVICE` calls (or graph patterns) and improve the efficiency of the query while securing the correct results. Our query planner is abstractly illustrated in 1.2. Briefly speaking, the first step is to analyze the `SERVICE` patterns of the query Graph, the second step is to use the estimated costs based on heuristics to search for the best plan and finally to rewrite the query based on this plan.

Having said that, remote query executions may frequently lead to unsuccessful results or failed query attempts. Since maintaining a reliable endpoint requires a significant additional cost (both in effort and in computer resources) that not all publishers are willing or able to pay. Notice also that HTTP servers are much more reliable than SPARQL endpoints. Actually, *availability* is the main bottleneck towards the success of the Semantic Web as a reliable technology. Buil-Aranda et al. [20] tested 427 public endpoints and found that their performance can vary by up to 3-4 orders of magnitude, while only 32.2% of public endpoints can be expected to have monthly uptimes of 99-100%. Therefore, it may be more reliable to directly retrieve the triples of a dereferenceable URI than retrieving the same triples by invoking a query against a remote endpoint (considering of course that the query requirements are satisfied). With this in mind in this thesis we also present the format extension implemented on `SPARQL-LD`, to enable querying HTML resources that contains JSON-LD, Microdata or Microformats. An example in Figure 1.3, shows a Web page with data about contact information, social profile links and logo image, that are in Microdata format, which can be extracted by the `SPARQL-LD`.

1.3 Contributions

In this context, the main contributions of this thesis are:

- We identify factors that affect query efficiency and propose a heuristics based static query reordering optimization technique on graph pattern restrictions,

[2] `https://query.wikidata.org/`

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbc:<http://dbpedia.org/resource/Category:>
SELECT Distinct ?journal ?journal_info WHERE {
  SERVICE <http://dbpedia.org/sparql> {              #538
      ?dbp_journal a dbo:AcademicJournal.
      ?dbp_journal dct:subject dbc:Toxicology_journals.
      ?dbp_journal owl:sameAs ?journal.
      ?dbp_journal rdfs:label ?label.
  }
  SERVICE <https://query.wikidata.org/sparql> {      #3
      ?journal wdt:P31 wd:Q5633421.
      ?journal wdt:P571 "1978-01-01"^^xsd:dateTime.
      ?journal wdt:P495 wd:Q145.
  }
  SERVICE ?journal {                                 #176
      ?journal ?p ?journal_info.
  }
}
```

Figure 1.1: SPARQL query against multiple endpoints (such as DBpedia and Wikidata) to retrieve information about resources classified as scientific journals.

by evaluating the syntactic variations of graph patterns, in order to execute close to optimal execution plans.

- We devise a parallel fetching technique of remote HTTP resources that significantly improves the performance of the SPARQL-LD in some circumstances.
- We showcase a caching optimization of SERVICE bindings retrieved from remote sources to increase the efficiency of the SPARQL-LD.
- We present an extended format support for SPARQL-LD over popular HTML formats i.e., embedded JSON-LD, Microformats and Microdata that are used extensively on the Web [44].
- We extensively evaluate the effectiveness of SPARQL-LD as of the proposed

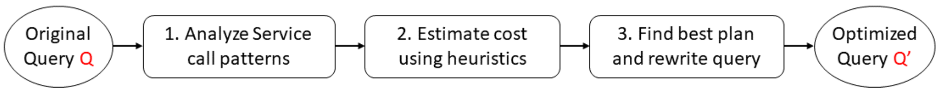
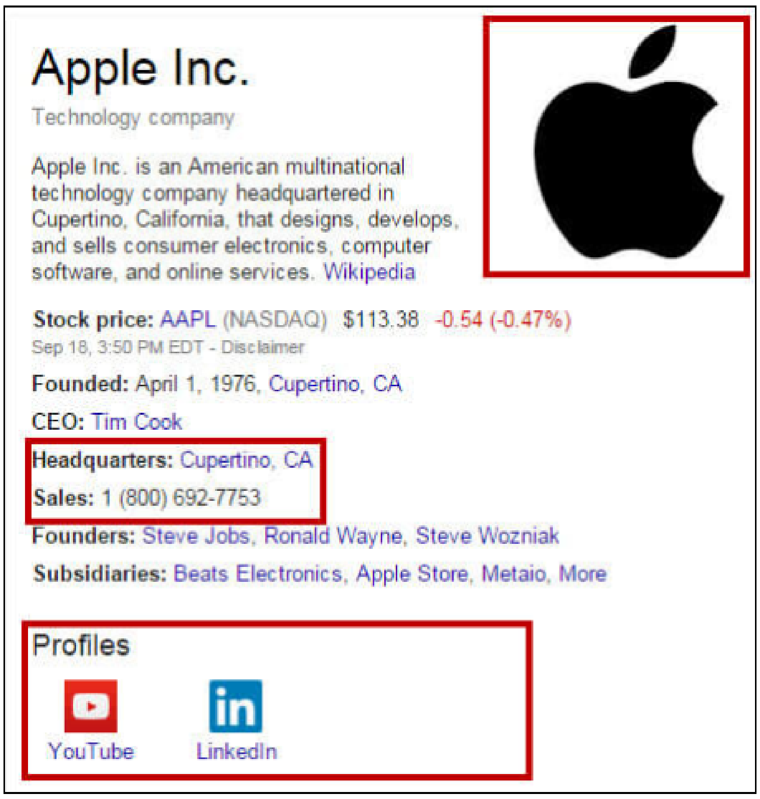

Original Query Q
1. Analyze Service call patterns
2. Estimate cost using heuristics
3. Find best plan and rewrite query
Optimized Query Q'

Figure 1.2: The basic steps of our query planner.



Apple Inc.

Technology company

Apple Inc. is an American multinational technology company headquartered in Cupertino, California, that designs, develops, and sells consumer electronics, computer software, and online services. Wikipedia

Stock price: AAPL (NASDAQ) $113.38 -0.54 (-0.47%)

Sep 18, 3:50 PM EDT - Disclaimer

Founded: April 1, 1976, Cupertino, CA

CEO: Tim Cook

Headquarters: Cupertino, CA

Sales: 1 (800) 692-7753

Founders: Steve Jobs, Ronald Wayne, Steve Wozniak

Subsidiaries: Beats Electronics, Apple Store, Metaio, More

Profiles

YouTube LinkedIn

Figure 1.3: An Web page using Microdata format.

optimizations techniques and offer more complete results over real datasets that are widely used such as dbpedia, wikidata and yago[3].

[3]https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/

1.4 Outline of Thesis

The rest of this thesis is organized as follows:

Chapter 2 discusses the context and describes related work and what distinguishes the current one in contrast to the known literature.

Chapter 3 presents `SPARQL-LD`, its functionality and the extended service operator and provides architectural and applicability details.

Chapter 4 details the optimization techniques proposed by this thesis such as static group graph pattern reordering optimizations using selectivity estimation and methods for parallelizing RDF data retrieval for efficient data caching.

Chapter 5 evaluates the performance of the `SPARQL-LD`. Then it reports and analyzes the experimental results over 3 real datasets and also introduces statistical metrics on datasets used.

Finally chapter 6 concludes the thesis and identifies directions for future research.

Chapter 2

Context and Related Work

2.1 Background

In this section, we discuss the importance of the Semantic Web and introduce the Resource Description Framework (RDF) data model [37] and its SPARQL [8] query language for querying RDF data. Finally we present the `SPARQL-LD` [24] and its infrastructure on which we base our optimizations.

2.1.1 Semantic Web, RDF and LOD

The vision of the Semantic Web is to publish and query knowledge on the Web in a semantically structured way. The idea of a Semantic Web was introduced to a wider audience by Berners-Lee in 2001 [16]. According to his vision, the traditional Web as a Web of Documents should be extended to a Web of Data where not only documents and links between documents, but any entity (e.g., a location or a person) and any relation between entities (e.g., isFatherof) can be represented on the Web. When it comes to realizing the idea of the Semantic Web, knowledge graphs (KGs) are currently seen as one of the most essential components. The term "Knowledge Graph" was coined by Google in 2012 and is intended for any graph-based knowledge base. We define a Knowledge Graph as an RDF graph.

The Resource Description Framework (RDF) is a graph-based data model recommended by W3C for publishing (linked) Web data on the Semantic Web.

RDF is based on the concept of resource which is everything that can be referred to through a Uniform Resource Identifier (URI). In particular, RDF builds on triples to relate URIs to others URIs, to constants called literals or to unknown values called blank nodes (which are similar to the notion of labelled nulls in incomplete databases). A triple is a statement ($s\ p\ o$) meaning that the subject s is described using the property p (a.k.a. predicate) by having the object value o. Formally, given $\mathcal{U}$, $\mathcal{L}$ and $\mathcal{B}$ denoting three (pairwise disjoint) sets of URIs, literals, and blank nodes respectively, a well-formed triple is a tuple ($s\ p\ o$) from $(\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{L} \cup \mathcal{B})$. In the following, we only consider well-formed triples.

A set of triples is an RDF graph, in which every triple $(s\ p\ o)$ corresponds to a directed edge labelled with $p$from the node labelled with $s$to the node labelled with o.

In this thesis, we focus on those KGs having specific aspects, for instance the KGs should be freely accessible and freely usable within the Linked Open Data (LOD) cloud (an example is shown in figure 2.1 [63]). Linked Data refers to a set of best practices [53] for publishing and interlinking structured data on the Web, defined by Berners-Lee [56]. Linked Open Data refers to the Linked Data which "can be freely used, modified, and shared by anyone for any purpose." The aim of the Linking Open Data is to publish RDF data sets on the Web and to interlink these data sets.

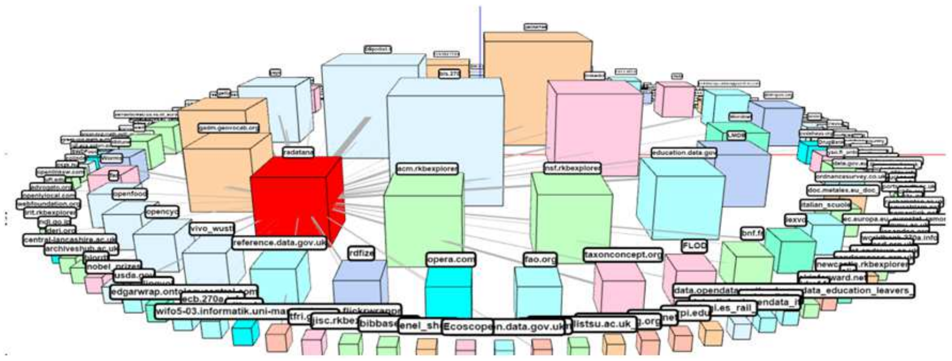


Figure 2.1: A 3D Visualization of LOD datasets.

For selecting the KGs for analysis, we regarded all datasets which fulfilled the above mentioned requirements. Based on that, we selected DBpedia, Wikidata, and YAGO as KGs for our comparison. In this thesis, we give a systematic overview of these KGs. Furthermore, we provide an evaluation information for users who are interested in using one of the mentioned KGs in a research or industrial setting, but who are inexperienced in which KG to choose for their concrete purposes.

2.1.2 SPARQL

SPARQL [51] is the W3C standard for querying RDF graphs. In this thesis, we consider the Basic Graph Pattern (BGP) queries of SPARQL, i.e., its conjunctive fragment allowing to express the core Select-Project-Join database queries. In such queries, the notion of triple is generalized to that of triple pattern $(s\ p\ o)$ from $(\mathcal{U} \cup \mathcal{B} \cup V) \times (\mathcal{U} \cup V) \times (\mathcal{U} \cup \mathcal{L} \cup \mathcal{B} \cup V)$, where V is a set of variables. The normative syntax of BGP queries is SELECT $?v_1$ ... $?v_m$ WHERE t_1 ... t_n where t_1, ..., t_n are triple patterns and $?v_1$, ..., $?v_m$ are distinguished variables occurring in $t_1 \cdots t_n$ which define the output of the query. Observe that repeating a variable among triple patterns is the way of expressing joins. In the following, we assume BGP queries which do not contain cartesian products. The evaluation of a query q, defined as SELECT $?v_1 \cdots ?v_m$ WHERE $t_1 \cdots t_n$, on an RDF graph G is: $eval(q) =$

$\{f(?v_1 \cdots ?v_m) | f : varbl(q) \to val(G)$ is a function s.t. $\{f(t_1), \cdots, f(t_n)\} \subseteq G\}$, with $varbl(q)$ the set of variables and blank nodes occurring in q, $val(G)$ the set of URIs, literals and blank nodes occurring in G, and f a function replacing any variable or blank node of q with its image in $val(G)$. By a slight abuse of notation, we denote by $f(t_i)$ the triple obtained by replacing the variables or blank nodes of the triple pattern t_i according to f. Observe that blank nodes do not play any particular role in queries, since (normative) query evaluation treats them as non-distinguished variables.

2.1.3 SPARQL-LD

This thesis is based on `SPARQL-LD`, which is an extension (actually a generalization) of SPARQL 1.1 Federated Query [9] that allows to *directly* and *flexibly* exploit this wealth of data. `SPARQL-LD` extends the applicability of the `SERVICE` operator (of SPARQL 1.1 Federated Query) enabling to query any HTTP Web source containing RDF data. This extension does not require the named graphs to have been declared, thus one can even fetch and query a dataset returned by a portion of the query (i.e., whose URI is derived at query execution time).

Such a functionality can motivate Web publishers to enrich their documents and digital libraries with RDF since it makes their data directly accessible via SPARQL without needing to set up and maintain an endpoint (e.g., they can just publish RDF dumps). The use of an endpoint for a dataset consisting of a small number of triples does not justify the provision effort. Maintaining a reliable endpoint requires a significant additional cost (both in effort and in computer resources) that not all publishers are willing or able to pay. Notice also that HTTP servers are much more reliable than SPARQL endpoints. Actually, *availability* is the main bottleneck towards the success of the Semantic Web as a reliable technology. Buil-Aranda et al. [20] tested 427 public endpoints and found that their performance can vary by up to 3-4 orders of magnitude, while only 32.2% of public endpoints can be expected to have monthly uptimes of 99-100%. Therefore, it may be more reliable to directly retrieve the triples of a dereferenceable URI than retrieving the same triples by invoking a query against a remote endpoint (considering of course that the query requirements are satisfied).

Fig. 2.2 shows a query that can be answered by `SPARQL-LD`. The query first accesses Europeana's [33, 36] SPARQL endpoint[1] for retrieving artists of works related to Renaissance (lines 2-3). Then, by querying the dereferenceable URI of each artist, the query retrieves and shows a description (in English) and an image of only those of Mannerist style (lines 4-6). Note that Europeana does not contain information about artist styles. Notice also that the artist URIs are derived at query execution time. One could also integrate in the same query data from any Web resource or Web Service that offers its data in a standard RDF or microdata format. As an example, consider that an online bookstore service exports its search

[1]`http://sparql.europeana.eu/`

results in microdata. Using `SPARQL-LD` one can directly access this service through SPARQL and find books about the artists returned by the two `SERVICE` patterns in the query of Fig. 2.2. Likewise, in the same query one could exploit a video service and find links of YouTube videos related to some of the artists.

Consequently, the functionality offered by `SPARQL-LD` can overcome limits of information integration, enrichment and exploitation. We will further discuss `SPARQL-LD` in Section 3.

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
PREFIX dct: <http://purl.org/dc/terms/subject>
PREFIX dbo: <http://dbpedia.org/ontology/abstract>
PREFIX foaf: <http://xmlns.com/foaf/0.1/depiction>
SELECT DISTINCT ?creator ?descr ?photo WHERE {
  SERVICE <http://sparql.europeana.eu/> {
    ?work dc:subject dbr:Renaissance .
    ?work dc:creator ?creator . }
  SERVICE ?creator {
    ?creator dct:subject dbc:Mannerist_painters .
    ?creator dbo:abstract ?descr .
    ?creator foaf:depiction ?photo FILTER(lang(?descr)="en") . } }
```

Figure 2.2: An example of a SPARQL query that can be answered by `SPARQL-LD`.

`SPARQL-LD` language was first demonstrated in a short (demo) paper by Fafalios et al. [23] that extended the SPARQL 1.1 Federated Query, by enabling to query any HTTP resource such as web pages containing RDFa and embedded turtle format or directly querying RDF files, e.g., rdf/xml, turtle and n3 formats. Furthermore his work includes two optimizations, i.e., index of know SPARQL endpoints and request-scope caching of fetched dataset optimization. Compared to this work in this thesis we extended the supported formats, by enabling to query any web page containing JSON-LD, Microdata and Microformats I & II. Moreover we further analyzed more optimization cases and present two more, i.e., parallel fetching of remote resources and query reordering optimization. Finally we have updated `SPARQL-LD` to support the newer version of Jena and ARQ.

`SPARQL-LD` was also published in the conference paper [24].

2.2 Related Work

In Subsection 2.2.1 we discuss the main methods for querying Linked Data, while in Subsection 2.2.2 we present similar optimization that `SPARQL-LD` can incorporate.

2.2.1 Basic Methods for Querying Linked Data

The approach that we propose is considered a method to execute queries over the Web of Linked Data. Such approaches can be classified in three main categories: *query federation*, *data centralization*, and *link traversal*.

The idea of *query federation* is to provide integrated access to distributed sources (RDF sources in our case) on the Web. For example, the DARQ engine [52] provides transparent query access to multiple SPARQL services by giving the user the impression that one single RDF graph is queried despite the real data being distributed on the Web. Similarly, the `SemWIQ` system [39] provides access to distributed RDF data sources using a mediator service that transparently distributes the execution of SPARQL queries without the need to specify the target endpoints. Given the need to address query federation, in 2013 the SPARQL W3C working group proposed a query federation extension for SPARQL 1.1 [9]. Buil-Aranda et al. [19] describe the syntax of that extension and formalize its semantics.

The idea of *data centralization* is to provide a query service over a collection of Linked Data copied (and probably transformed) from different sources on the Web. Such a collection of sources is usually called "Warehouse". There are *domain independent* warehouses like Sindice [46] and SWSE [34], but also *domain specific* like the MarineTLO-based Warehouse [60]. Such approaches are quite distant to the context of our work since they require the data to exist in a single repository, but they can significantly benefit from the functionality offered by `SPARQL-LD`. For instance, a query service over such a repository can support `SPARQL-LD` and offer the ability to also integrate (during query execution) data coming from online RDF sources (like the example in Fig. 2.2)

Link traversal approaches exploit the Linked Data principles for discovering data related to URIs given in the query. For instance, the work in [29,30] discovers data that might be relevant for answering a query, by following RDF links between data sources based on URIs in the query and in partial results. Specifically, the URIs are resolved over the HTTP protocol into RDF data which is continuously added to the queried dataset using an iterator-based pipeline. `Diamond` [42] is a similar in spirit query engine to evaluate SPARQL queries on distributed RDF data where, as a query is being evaluated, additional Linked Data can be identified by exploiting dereferenceable URIs. Finally, LDQL [32] is a declarative language to query Linked Data which is also based on link traversal. LDQL separates query components for selecting query-relevant regions of Linked Data, from components for specifying the query result that has to be constructed from the data in the selected regions.

2.2.2 Methods for optimizing SPARQL queries

Works on query planning optimizations techniques for SPARQL. *Works on query planning optimizations techniques for SPARQL* [21] includes but not limited to: a)query rewriting based on statistics optimizations, b)selectivity based optimizations (heuristics), c)mixed strategy for query optimizations (hybrid), d)fedex framework and e)graph traversal algorithm for SPARQL query optimizations.

Heuristics based. The idea of static query optimization (which is the main focus of this thesis), i.e. join ordering optimization of triple patterns before query evaluation is to find an execution plan which will return results sets the fastest, before executing the query, which is typically coupled with the use of heuristics and summaries (e.g., statistics) about the data being queried.

One such notable work is Tsialiamanis et al. [59], proposes a set of heuristics on triple patterns for query optimization without using statistics, supported by their experiments. More specific the method which they represent their query statements are by using basic triple patterns to build a graph that is composed of nodes and edges as query planning space. They built a graph called variable graph, which only considers the variables appearing in triple patterns, and then they assign a weight to these variables by using the number of their occurrence in the SPARQL query. This representation is related directly to their cost estimation model, which aims to find maximum weight independent sets of the variable graph. In the same context Song et al. [57] provides a heuristic-based approach by extending the work in [59]. They provide more heuristics and expand the evaluation cost by taking into consideration more expressions such as *filter*. Their query planning methods are implemented within Corese and the system is evaluated using BSBM benchmark.

Query rewriting based on statistics optimizations. An important aspect of the execution time of queries is heavily influenced by the number of joins necessary to find the results of the query. Therefore, the goal of query optimization is (among other things) to reduce the number of joins required to evaluate a query. Such optimizations typically focus on histograms based selectivity estimation of query conditions. Piatetsky introduce in [49] the concept of selectivity estimation of a condition. Estimation of conditions are often supported by histogram distributions of attribute values. Related to the Semantic Web, Perez analyze in [48] the semantics and complexity of SPARQL. Harth [28] investigate the usage of optimized index structures for RDF. The authors argue that common RDF infrastructures do not support specialized RDF index structures. The index proposed by the author supports partial keys and allows selectivity computation for single triple patterns. Hartig et al. [31] present a SPARQL query graph model (SQGM) which supports all phases of query processing, especially query optimization and a set of operators to model the SPARQL operations. The author provides transformation and rewriting rules to SPARQL on Jena to exploit the fast path algorithm more often.

Similarly Neumann et al. [45] present the RDF-3x a native-RDF system that relies heavily on the use of indexes to process SPARQL queries over compressed RDF triples. In particular, triples are compressed by lexicographically sorting them and storing only the changes between them. RDF-3X builds a clustered B+tree index with composite keys over every possible collation order of triple components. Furthermore, RDF-3X uses aggregated indexes for each of the three possible pairs of triple components and in each collation order (sp, so, ps etc.). Each index stores the two columns of a triple on which it is defined and an aggregated count that denotes the number of occurrences of the pair in the set of triples. Aggregated indexes that are organized in B+-trees, are much smaller than the full-triple indexes and are used to avoid decompressing duplicate triples in the final query results. In addition, RDF-3X builds all three onevalue indexes that hold for every RDF constant the number of its occurrences in the dataset. This work also analyzes join ordering optimizations by using sorted index lists and statistics about counters of frequent predicate-sequences paths that are potential join patterns. Despite the exhaustive indexing employed by RDF-3X, the size of the indexes does not exceed the size of the dataset thanks to the compression scheme. Furthermore this work relies on a cost model to estimate the number of intermediate results based on statistics and makes use of histograms to precompute the most frequent paths for query patterns. In contrast, our heuristic-based SPARQL planner produces plans using solely the heuristics and without statistics.

Schmidt et al. [54] discusses the complexity of SPARQL evaluation, focusing on Optional operator and analyze rewriting techniques. In particular this work study the set of equivalences over the SPARQL algebra as well as well known to relational algebra rewriting rules. Moreover, this work proposes an approach to semantic query optimization, based on the classical chase algorithm.

Hybrids based. Some systems take advantage of both statistics to evaluate the cost estimation as well some provided heuristics based cost estimations. One such work is from Stocker in [58] that provides a static basic graph pattern optimization based on triple patterns variable counting using a main memory graph implementation as well as analyzing the heuristics of joined triple patterns using a probabilistic framework. They evaluate each triple pattern using a formula that provides a selectivity score ranging from 0 to 1, thus being able to reorder the triple patterns based on this score. Moreover this work uses a histogram, such methods are widely used for storing summary data, this practice maintains relative high accuracy but cost much time and storage space. Another similar work is described by Gropper et al. [27] where the authors provide a brief optimization on joined triple pattern reordering by counting new variables of triple patterns. In addition this work measures the result size of join operands if the above method is inclusive and reorder the execution plan accordingly. They also make use of indexing optimizations coupled with statistics about the knowledge bases used.

The work by Montoya in [43] contains *query reordering* optimization through decomposing queries in sub-queries according to joins (or UNIONS) in order to assign an endpoint to each one heuristically (using the namespace and running ask queries) by evaluating each IRI predicate. Furthermore calls to endpoints are sent to retrieve the number of expected returned triples, for each sub query and use this information to execute their query plan and reorder the sub queries accordingly. Liu et al. [41] proposes a query planner that uses basic triple patterns to build a graph that is composed of nodes and edges as query planning space. But they differentiated by using nods as normal vertices and triple vertices, the difference is that the former refers to the variables with constraints in a triple pattern and

Table 2.1: Comparison between notable works on query planning.

Author	Sparql Representation	Cost Estimation
Tsialiamanis 2012	Pattern variable graph	Heuristic-based
Huang 2010	-	Stats-based
Liu 2010	SPARQL query graph	Stats-based
Stocker 2008	Basic pattern graph	Hybrid
Neumann 2008	Basic pattern graph	Stats-based
Our Work	Service-Graph pattern	Heuristic-based

the latter refers to the whole triple pattern, so forth the normal edges and triple edges. This approach maintains not only the connections between triple patterns but also the relations within the triple patterns themselves.

Most of the above works don't discusses how query plans (i.e., join orders and join variables) are found as we do in our work. In the work by Vidal et. al. [62] RDF triples are stored in a large triple table and a set of physical operators are proposed for efficiently implementing starshaped queries. In this work, a randomized cost-based optimization strategy is adopted to determine the most cost-effective plan among a set of execution plans of any shape (bushy, left deep etc.). The cost-based optimizer uses statistics about the size of properties, and the selectivity of subjects and objects to determine the most prominent star-shaped joins. In our work, we are able to produce near to optimal plans without the use of any statistics, we rely on the proposed heuristics. Another work by Huang et al. [35] focuses on join estimation for star queries and chain queries patterns with correlated properties and precomputed statistics. First they use the Bayesian networks for star queries to compactly represent the joint probability distribution over values of correlated properties and then chain histogram for star queries, which can obtain a good balance between the estimation accuracy and space cost. Similarly Pichler in [50] discusses about the complexity about equivalence on Join and Optional operators as well as on union and projection operations. Furthermore Bernstein in [17] proposes OptARQ an optimization framework based on selectivity estimation to reduce the intermediate result sets of triple patterns. This optimization is based on static query rewriting rules and leverages histograms based models for its selectivity estimation on triple patterns. Although this work uses statistical information about the underlying ontological resources in order to get an estimation and rank the patterns and thus is unsuitable for our work. Finally they make use of filter rewriting rules reduce the intermediate result sets.

In this thesis, we will adopt a method similar to [59] and [58], which propose a set of heuristics based on the sparql syntax to reorder triple patterns. However in this thesis we evaluate graph patterns (i.e., `SERVICE` calls) instead of triple patterns and extend those heuristics. In table 2.1 we provide a comparison between notable works that focus on heuristic based cost estimation as well as statistic based cost estimation.

Works on parallelization techniques for SPARQL. *Works on parallelization techniques for SPARQL* and parallel solutions include works such as the FedX framework et al. [55] that allows for virtual integration of heterogeneous LOD sources into a federation, on-demand without preprocessing. FedX creates left linear plans of sub-queries, using a rule-based system and can each be answered exclusively by existing endpoints (or exclusive groups). Then with the use of bounded variables the framework decides the query join order which is evaluated in a block-nested loop fashion to reduce the number of source requests. One of the main contribution of this work is their use of executing parallel queries using an exclusive group technique.

Query equivalence works. It is crucial in query optimization to always retrieve the correct results. Some related works on query equivalence include: the work of Angles et al. [15] which focus on rewriting some SPARQL operations such as unsafe filter operations or specific differences between patterns (such as a transformation from a **minus** pattern to a combination of optional and filter ones). Also the work from Kochut on [38] provides SPARQLeR, a SPARQL query validator that can transform queries to SPARQL algebra and adding support to semantic path queries.

Chapter 3

SPARQL-LD: Functionality and Examples

In this chapter we present the system `SPARQL-LD` previously described in [24] as well as the format extension that is implemented in this thesis.

3.1 Architecture

Although the majority of SPARQL implementations requires the data to be available in advance (in main memory or in a repository), the specification of SPARQL allows to directly query an RDF dataset accessible on the Web (in a standard format) and identifiable by an URI through the operators `FROM`/`FROM NAMED` and `GRAPH`. However, this has an important limitation: it requires knowing *in advance* the URI of the dataset and having declared it in the `FROM NAMED` clause. Thus, a URI coming from partial results (that get bound after executing an initial query fragment) cannot be used in the `GRAPH` operator as the dataset to run a portion of the query. Furthermore, although RDFa [7] and JSON-LD [1] are W3C standards that are exploited by an ever-increasing number of publishers, we have not managed to find a SPARQL implementation that can directly query such RDF data. In addition, using the `SERVICE` operator of SPARQL 1.1 Federated Query [9], we can invoke a portion of a query against a remote RDF repository. However, `SERVICE` requires the URI to be the address of a SPARQL endpoint, thus one cannot exploit this operator for querying RDF data accessible on the Web but not available through an endpoint.

Extended `SERVICE` definition. The SPARQL 1.1's `SERVICE` operator (`SERVICE` a P) is defined (in [19]) as a graph pattern P evaluated in the SPARQL endpoint specified by the URI a, while (`SERVICE` $?X$ P) is defined by assigning to the variable $?X$ all the URIs (of endpoints) coming from partial results, i.e. that get bound after executing an initial query fragment. The idea behind `SPARQL-LD` is to enable the evaluation of a graph pattern P not absolutely in a SPARQL endpoint a, but generally in an RDF graph G_r specified by a Web Resource r. Thus, now a URI given to the `SERVICE` operator can also be the dereferenceable URI of a resource, the Web page of an entity (e.g., of a person), an ontology (OWL), Turtle, or N3 file, etc. An example is shown in Fig. 3.1 In case the URI is not the address of a SPARQL endpoint, the RDF data that may exist in the resource are fetched at real-time and queried for the graph pattern P.

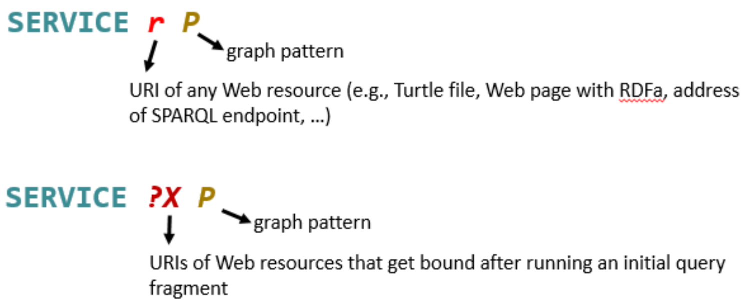

SERVICE r P
graph pattern
URI of any Web resource (e.g., Turtle file, Web page with RDFa, address of SPARQL endpoint, ...)
SERVICE ?X P
graph pattern
URIs of Web resources that get bound after running an initial query fragment

Figure 3.1: Sparql-LD extended Service function.

`SPARQL-LD` is a generalization of SPARQL in the sense that every query that can be answered by the original SPARQL can be also answered by `SPARQL-LD`. Specifically, if the URI given to the `SERVICE` operator corresponds to a SPARQL endpoint, then it works exactly as the original SPARQL (the remote endpoint evaluates the query and returns the result). Otherwise, instead of returning an error (and no bindings), it tries to fetch and query the triples that may exist in the given resource.

Moreover, the execution of a `SERVICE` pattern may fail due to several reasons. For instance, if the given URI corresponds to an endpoint, the endpoint may return an error to the query, while if the URI corresponds to a Web page, the page may be down. In such cases, the invoked query containing a `SERVICE` pattern normally fails as a whole. In SPARQL 1.1 Federated Query, queries may explicitly allow failed `SERVICE` requests with the use of the `SILENT` keyword which indicates that possible errors encountered while accessing a remote SPARQL endpoint should be ignored and a single solution with no bindings should be returned. In `SPARQL-LD`, the presence of the `SILENT` keyword has exactly the same functionality, i.e. it allows failed `SERVICE` requests.

3.2 Extended Functionality

`SPARQL-LD` has been implemented using Apache Jena [3] framework. Jena is an open source Java framework for building Semantic Web applications. Specifically, we have extended Jena 3.3 ARQ component[1]. ARQ is a query engine for Jena

[1]`http://jena.apache.org/documentation/query`

that supports SPARQL 1.1. The implementation is available as open source[2]. An endpoint that supports SPARQL-LD is publicly available[3].

The implementation can be described through the following process (depicted at Figure 3.2): we first check if the URI corresponds to a SPARQL endpoint by submitting the ASK query "ASK {?x ?y ?z}". In case we get a valid answer, we continue just like the default query federation approach, i.e. the corresponding graph pattern (query) is submitted to the endpoint. In case we do not get a valid answer, it means that the URI is not the address of an endpoint. Then, we read the *content type* header field of the URI by opening an HTTP connection and setting the value application/rdf+xml to the ACCEPT request property. Now, according to the returned content type, we fetch and query the corresponding triples.

Since structure data that are embedded in HTML pages has seen an increase in use, we enabled SPARQL-LD to query more encoding formats that are embedded in web pages like microdata, microformats and JSON-LD. For the case of HTML Web pages(the content type is text/html or application/xhtml+xml), we try to fetch and query the structured data that may be embedded in the Web page as RDFa, JSON-LD, Microformats or Microdata. For this purpose the a modified version of Apache - ANY23 framework [2] was incorporated in SPARQL-LD . The process is separated in three steps , the first step was to use Apache - ANY23 mime detector (Apache Tika) to identify the format type of the data in the web page. Then using the information that was retrieved, the appropriate resource data extractor was selected that will retrieve the required web embedded resources and transform them in RDF triples (if they are not already). Finally the data are loaded in our SPARQL-LD to enable the refined query process to take place. If the Web page does not contain any RDF data, the query returns no bindings. The implementation allows also to read and query N3, TURTLE, RDF\XML and JSON-LD files.

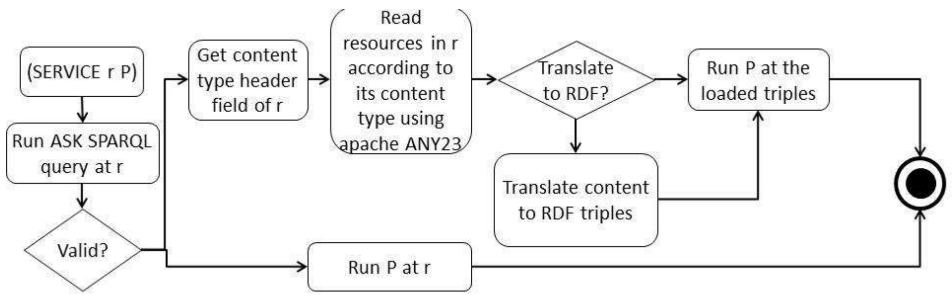

(SERVICE r P)
Run ASK SPARQL query at r
Valid?
Get content type header field of r
Read resources in r according to its content type using apache ANY23
Translate to RDF?
Run P at the loaded triples
Translate content to RDF triples
Run P at r

Figure 3.2: SPARQL-LD implementation process.

Here we give example queries that demonstrate the functionality offered by SPARQL-LD.

[2]https://github.com/fafalios/sparql-ld
[3]http://users.ics.forth.gr/~fafalios/sparql-ld-endpoint

3.2.1 Querying dynamically-created RDF data.

X-Link [22] is a Linked Data-based Named Entity Extraction (NEE) framework which can export the result of the NEE process in RDF using the Open NEE model [22]. An X-Link Web service configured for the marine domain is publicly available (http://139.91.183.72/x-link-marine). This service can identify names of several types of entities in a given Web document and link them to DBpedia [40] resources. For instance, we can request to perform NEE with *fishes* and *countries* as the entities of interest at the Web page: "http://www.hawaii-seafood.org/wild-hawaii-fish" and get the results in the default RDF/XML format, with the following request:

http://139.91.183.72/x-link-marine/api?categories=fish;country&link=1
&url=http://www.hawaii-seafood.org/wild-hawaii-fish

Using the proposed extension, one can exploit the APIs of such services directly through SPARQL. For instance, Figure 3.3 depicts a query that *parameterizes* and *calls* the above annotation service *at query execution time* (the namespaces have been omitted to save space). The query first retrieves Web pages related to the fish genus Thunnus by querying its dereferenceable URI (lines 2-3). Then, it calls the X-Link service for identifying names of *fishes* and *countries* in the retrieved Web pages (lines 4-6), and for each detected entity the query retrieves (and shows) its name, its category and its number of occurrences in the Web pages (lines 7-9). Finally, the entities are ordered by the number of occurrences in descending order (line 10). Thus, using this functionality, one can annotate a large corpus of Web documents and get the results in RDF, by simply writing and submitting a SPARQL query. Note also that this query can be answered by any endpoint that implements the proposed extension (independently of the data stored in the endpoint).

```
 1 SELECT DISTINCT ?detectedEntity ?categoryName (count(?position) as ?NumOfOccurrences) WHERE {
 2   SERVICE <http://dbpedia.org/resource/Thunnus> {
 3     dbr:Thunnus dbo:wikiPageExternalLink ?page }
 4   VALUES ?templ { <http://83.212.107.202/x-link-marine/api?categories=fish;country&url=PAGE> }
 5   BIND(REPLACE(str(?templ), "PAGE", str(?page), "i") as ?x) BIND(URI(?x) as ?serv)
 6   SERVICE ?serv {
 7     ?annot oa:hasBody ?ent .
 8     ?ent oae:regardsEntityName ?detectedEntity ; oae:position ?position .
 9     ?ent oae:belongsTo ?category . ?category rdfs:label ?categoryName }
10 } GROUP BY ?detectedEntity ?categoryName ORDER BY DESC(?NumOfOccurrences)
```

Figure 3.3: Example of a SPARQL query that parameterizes and calls an annotation service at query execution time.

3.2.2 Querying RDFa.

Using SPARQL-LD one could previously query data in RDFa format that are embedded in Web pages. Such RDF data is directly available through SPARQL.

For example, the query in Figure 3.4 returns all co-authors together with their publications. The list of co-authors is obtained by querying the RDF data that is embedded in the Web page, as shown in fig. 3.5 (lines 2-4), while their names and publications are obtained by querying the dereferenceable URI of each co-author (lines 5-7). Notice that the author URIs are derived at *query execution time*. A part of the fetched data is depicted in 3.8.

```
1 SELECT DISTINCT ?authorName ?paper WHERE {
2   SERVICE <http://users.ics.forth.gr/~fafalios/> {
3     ?p <http://purl.org/dc/terms/creator> ?author
4     FILTER(?author != <http://dblp.l3s.de/d2r/resource/authors/Pavlos_Fafalios>) }
5   SERVICE ?author {
6     ?author <http://xmlns.com/foaf/0.1/name> ?authorName .
7     ?paper <http://purl.org/dc/elements/1.1/creator> ?author.
8   }
9 }
```

Figure 3.4: Example of a SPARQL query that reads and queries RDF data embedded in a Web page (as RDFa) at query execution time.

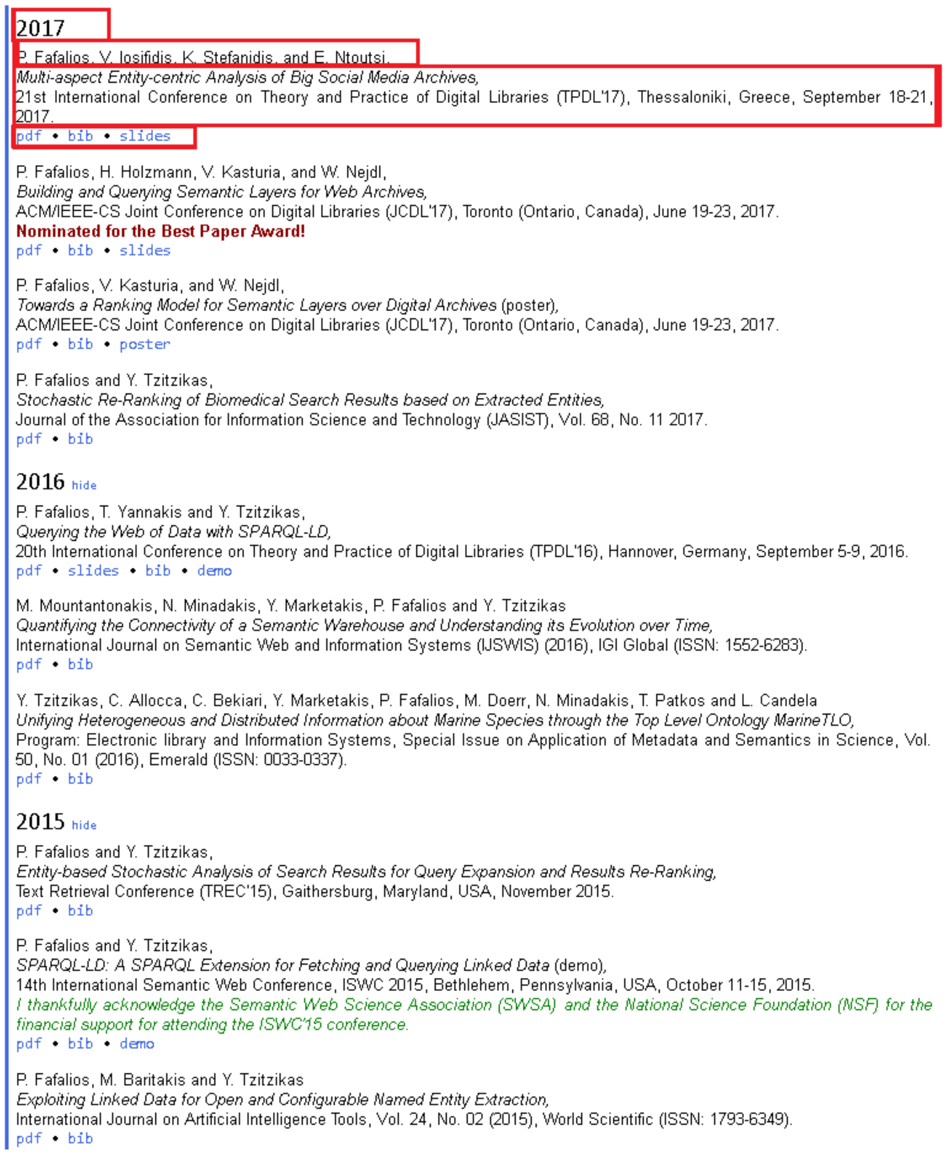
3.2.3 Querying JSON-LD embedded in HTML.

Using the proposed extension, JSON-LD data that are embedded in HTML pages through java-scripts are directly available through SPARQL.

For example, the query in Figure 3.6 returns the URL, logo and the available contact info from the web page embedded JSONLD script 3.9. The script that was retrieved is depicted in 3.10. Another example is also depicted in 3.7 that returns information from a Google search result. The actual JSON-LD script is shown in 3.11.

```
SELECT ?url ?logo (Str(?phone) as ?phonenum) WHERE {
 SERVICE <http://www.popsugar.com/> {
          ?jlitem <http://schema.org/logo> ?logo;
          <http://schema.org/url> ?url;
              <http://schema.org/contactPoint>/<http://schema.org/telephone> ?phon
 }
}
```

Figure 3.6: Example of a SPARQL query that reads and queries JSON-LD data embedded in a Web page at query execution time.



2017

P. Fafalios, V. Iosifidis, K. Stefanidis, and E. Ntoutsi,
Multi-aspect Entity-centric Analysis of Big Social Media Archives,
21st International Conference on Theory and Practice of Digital Libraries (TPDL'17), Thessaloniki, Greece, September 18-21, 2017.
pdf • bib • slides

P. Fafalios, H. Holzmann, V. Kasturia, and W. Nejdl,
Building and Querying Semantic Layers for Web Archives,
ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'17), Toronto (Ontario, Canada), June 19-23, 2017.
Nominated for the Best Paper Award!
pdf • bib • slides

P. Fafalios, V. Kasturia, and W. Nejdl,
Towards a Ranking Model for Semantic Layers over Digital Archives (poster),
ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'17), Toronto (Ontario, Canada), June 19-23, 2017.
pdf • bib • poster

P. Fafalios and Y. Tzitzikas,
Stochastic Re-Ranking of Biomedical Search Results based on Extracted Entities,
Journal of the Association for Information Science and Technology (JASIST), Vol. 68, No. 11 2017.
pdf • bib

2016 hide

P. Fafalios, T. Yannakis and Y. Tzitzikas,
Querying the Web of Data with SPARQL-LD,
20th International Conference on Theory and Practice of Digital Libraries (TPDL'16), Hannover, Germany, September 5-9, 2016.
pdf • slides • bib • demo

M. Mountantonakis, N. Minadakis, Y. Marketakis, P. Fafalios and Y. Tzitzikas
Quantifying the Connectivity of a Semantic Warehouse and Understanding its Evolution over Time,
International Journal on Semantic Web and Information Systems (IJSWIS) (2016), IGI Global (ISSN: 1552-6283).
pdf • bib

Y. Tzitzikas, C. Allocca, C. Bekiari, Y. Marketakis, P. Fafalios, M. Doerr, N. Minadakis, T. Patkos and L. Candela
Unifying Heterogeneous and Distributed Information about Marine Species through the Top Level Ontology MarineTLO,
Program: Electronic library and Information Systems, Special Issue on Application of Metadata and Semantics in Science, Vol. 50, No. 01 (2016), Emerald (ISSN: 0033-0337).
pdf • bib

2015 hide

P. Fafalios and Y. Tzitzikas,
Entity-based Stochastic Analysis of Search Results for Query Expansion and Results Re-Ranking,
Text Retrieval Conference (TREC'15), Gaithersburg, Maryland, USA, November 2015.
pdf • bib

P. Fafalios and Y. Tzitzikas,
SPARQL-LD: A SPARQL Extension for Fetching and Querying Linked Data (demo),
14th International Semantic Web Conference, ISWC 2015, Bethlehem, Pennsylvania, USA, October 11-15, 2015.
I thankfully acknowledge the Semantic Web Science Association (SWSA) and the National Science Foundation (NSF) for the financial support for attending the ISWC'15 conference.
pdf • bib • demo

P. Fafalios, M. Baritakis and Y. Tzitzikas
Exploiting Linked Data for Open and Configurable Named Entity Extraction,
International Journal on Artificial Intelligence Tools, Vol. 24, No. 02 (2015), World Scientific (ISSN: 1793-6349).
pdf • bib

Figure 3.5: Web page using RDFa format.

```
SELECT ?title ?logo ?searchTerm WHERE {
 SERVICE <https://www.google.gr/search?q=patrick+coombe+book&oq=patrick+coombe+boo
  ?jlitem <http://purl.org/dc/terms/title> ?title;
          <http://www.w3.org/1999/xhtml/vocab#icon> ?logo;
                     <http://www.w3.org/1999/xhtml/vocab#shortcut> ?searchTerm.
  }
}
```

Figure 3.7: Another example of a SPARQL query that reads and queries JSON-LD data embedded in a Web page at query execution time.

3.2.4 Querying Microdata embedded in HTML5.

With `SPARQL-LD` we can use SPARQL queries on web pages that contain data in Microdata format embedded in HTML5 pages.

For example, the query in Figure 3.12 returns all the customers feedback about their travel agency, such as rating, message and details about the vacation. The web page has enhanced its resources with Microdata, as show in fig. 3.13 and fig. 3.14.

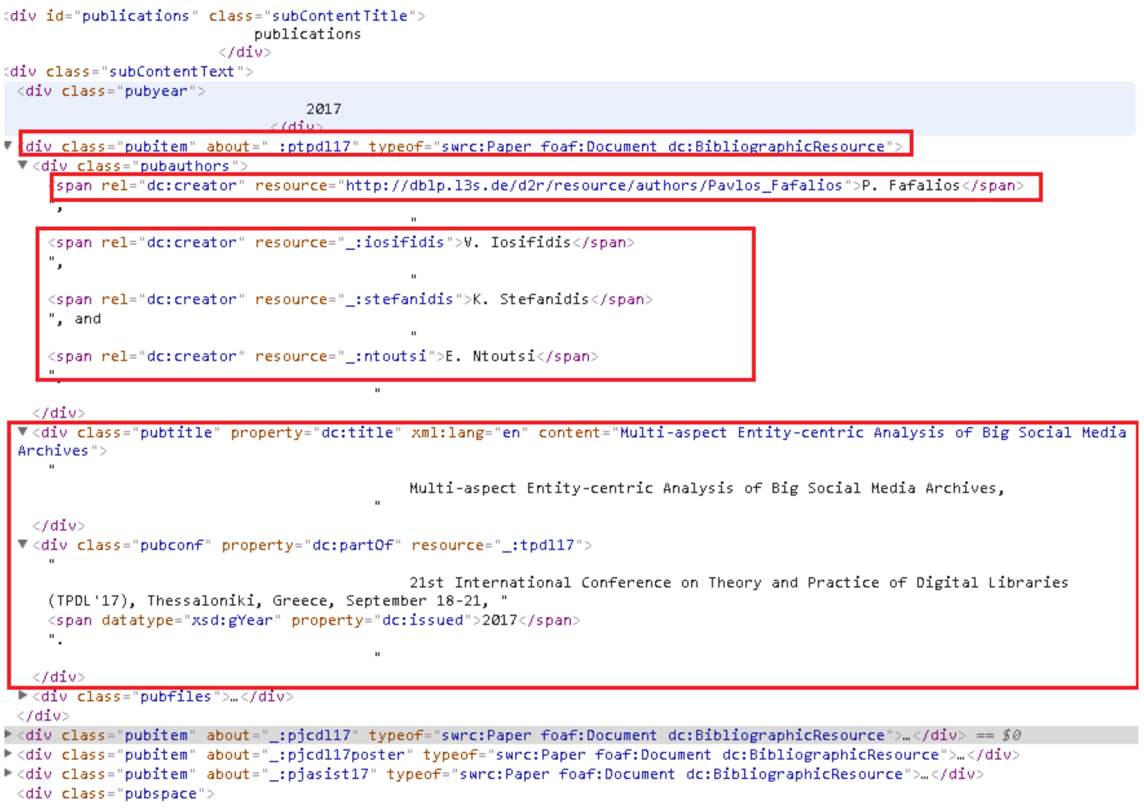
```
SELECT * WHERE {
 SERVICE <http://holidayplace.co.uk/about/customer-feedback> {
    ?s <http://data-vocabulary.org/Review/summary> ?description;
      <http://data-vocabulary.org/Review/bestRating> ?rating;
      <http://data-vocabulary.org/Review/itemreviewed> ?vacationInfo.
   }
}
```

Figure 3.12: Example of a SPARQL query that can execute queries over web pages that contain resources in Microdata format.

3.2.5 Querying Microformats embedded in HTML.

Another use of the proposed extension is that, we can use SPARQL queries on Microformats I & II that are embedded in HTML pages. Microformats is the simplest way to markup structured information in HTML, where Microformats2 improves ease of use and implementation for both authors (publishers) and developers. Such formats include but are not limited to HGEO, HCARD, HEVENT, HPRODUCT, HLOCATION and HENTRY, that involve markup information about specific people, locations, images, dates, blogspots or even events.

For example, the query in Figure 3.15 returns all contact information from the web page, about the author, such as name, surname and description of the web



```
<div id="publications" class="subContentTitle">
                           publications
                         </div>
<div class="subContentText">
 <div class="pubyear">
                                    2017
                              </div>
▼<div class="pubitem" about="_:ptpdl17" typeof="swrc:Paper foaf:Document dc:BibliographicResource">
 ▼<div class="pubauthors">
    <span rel="dc:creator" resource="http://dblp.l3s.de/d2r/resource/authors/Pavlos_Fafalios">P. Fafalios</span>
    ",
                                        "
    <span rel="dc:creator" resource="_:iosifidis">V. Iosifidis</span>
    ",
                                        "
    <span rel="dc:creator" resource="_:stefanidis">K. Stefanidis</span>
    ", and
                                        "
    <span rel="dc:creator" resource="_:ntoutsi">E. Ntoutsi</span>
    "
                                     "
   </div>
 ▼<div class="pubtitle" property="dc:title" xml:lang="en" content="Multi-aspect Entity-centric Analysis of Big Social Media
 Archives">
    "
                                          Multi-aspect Entity-centric Analysis of Big Social Media Archives,
                                     "
   </div>
 ▼<div class="pubconf" property="dc:partOf" resource="_:tpdl17">
    "
                                          21st International Conference on Theory and Practice of Digital Libraries
    (TPDL'17), Thessaloniki, Greece, September 18-21, "
    <span datatype="xsd:gYear" property="dc:issued">2017</span>
    ".
                                     "
   </div>
 ▶<div class="pubfiles">…</div>
 </div>
▶<div class="pubitem" about="_:pjcdl17" typeof="swrc:Paper foaf:Document dc:BibliographicResource">…</div> == $0
▶<div class="pubitem" about="_:pjcdl17poster" typeof="swrc:Paper foaf:Document dc:BibliographicResource">…</div>
▶<div class="pubitem" about="_:pjasist17" typeof="swrc:Paper foaf:Document dc:BibliographicResource">…</div>
 <div class="pubspace">
```

Figure 3.8: Fetched RDFa data.

page 3.16, that are in HCARD format. The returned data can be shown in fig. 3.17.

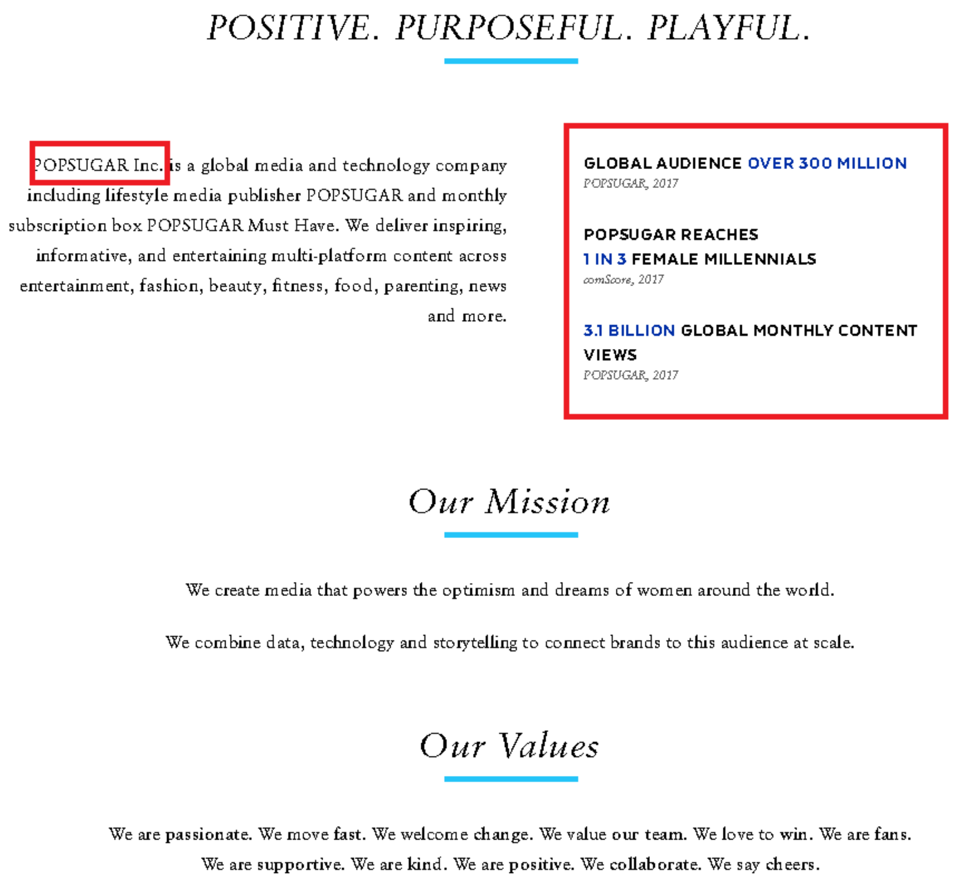
```
SELECT ?name ?subject WHERE {
  SERVICE <http://cold32.com/about-the-author-and-contact.htm> {
   ?jlitem <http://vocab.sindice.net/any23#dcterms.subject> ?subject;
            <http://vocab.sindice.net/any23#web_author> ?name.
}
```

Figure 3.15: Example of a SPARQL query that reads and queries Microformat data embedded in a Web page at query execution time.

s

3.3 Current Applications of SPARQL-LD

SPARQL-LD is a framework that can be used (and extended) by other applications according to their needs, allowing its exploitation in a plethora of contexts and



POSITIVE. PURPOSEFUL. PLAYFUL.

POPSUGAR Inc. is a global media and technology company including lifestyle media publisher POPSUGAR and monthly subscription box POPSUGAR Must Have. We deliver inspiring, informative, and entertaining multi-platform content across entertainment, fashion, beauty, fitness, food, parenting, news and more.

GLOBAL AUDIENCE OVER 300 MILLION
POPSUGAR, 2017

POPSUGAR REACHES
1 IN 3 FEMALE MILLENNIALS
comScore, 2017

3.1 BILLION GLOBAL MONTHLY CONTENT VIEWS
POPSUGAR, 2017

Our Mission

We create media that powers the optimism and dreams of women around the world.

We combine data, technology and storytelling to connect brands to this audience at scale.

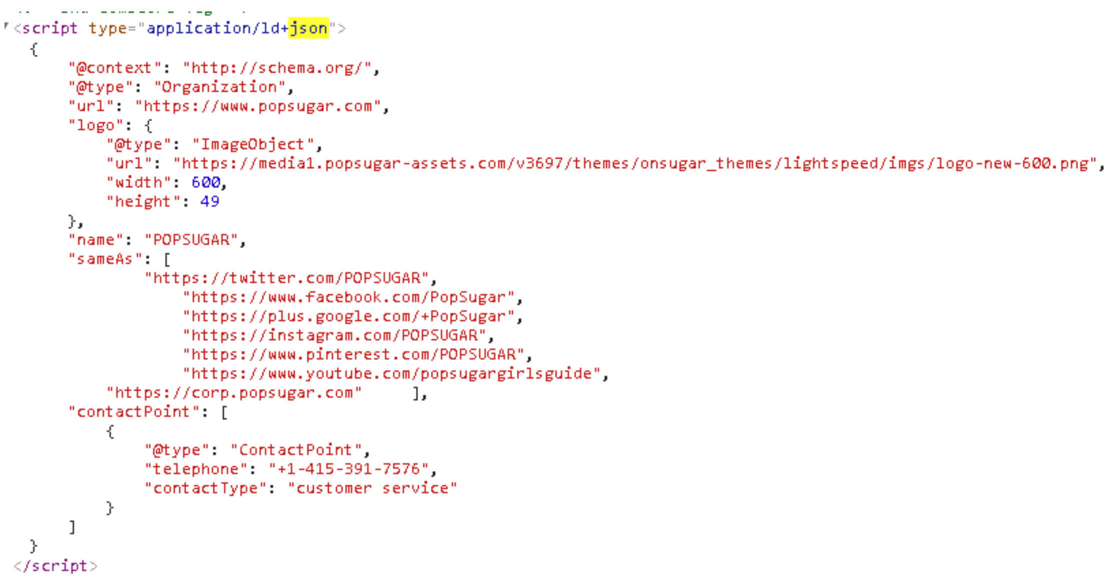
Our Values

We are passionate. We move fast. We welcome change. We value our team. We love to win. We are fans. We are supportive. We are kind. We are positive. We collaborate. We say cheers.

Figure 3.9: Web page using JSON-LD format.

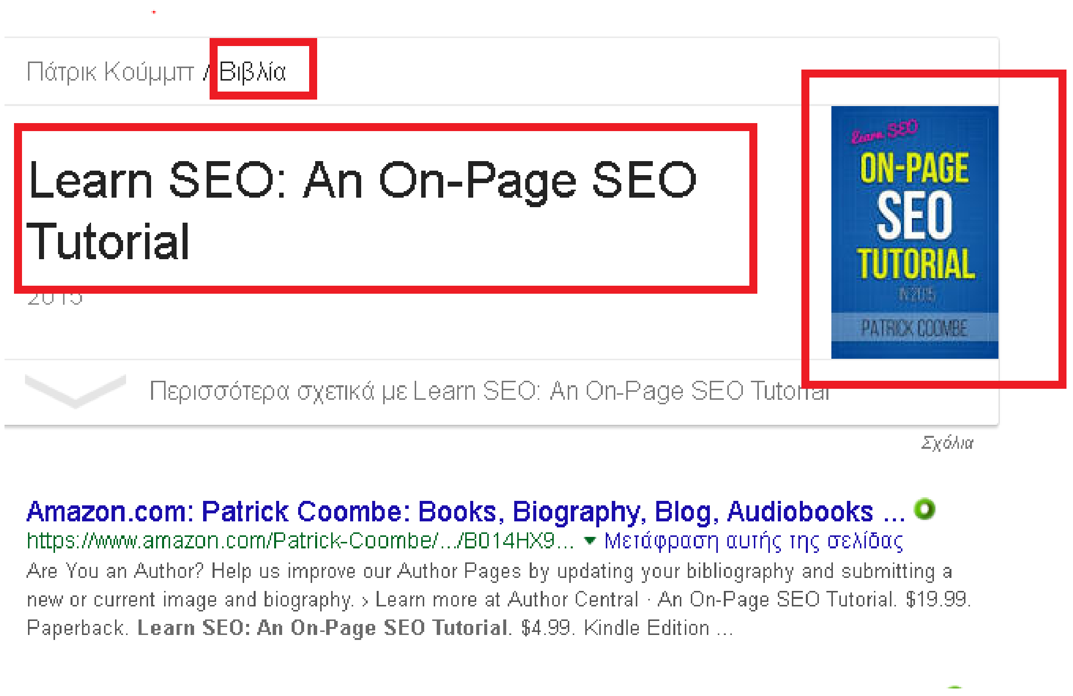
application scenarios. Specifically, `SPARQL-LD` can be used as a:

- Java Library which can be integrated in the code of the intended application.
- Web Application that can receive submissions and return the outcomes of the analysis.
- Web Service which can be used through a REST API.



```
<script type="application/ld+json">
  {
      "@context": "http://schema.org/",
      "@type": "Organization",
      "url": "https://www.popsugar.com",
      "logo": {
          "@type": "ImageObject",
          "url": "https://media1.popsugar-assets.com/v3697/themes/onsugar_themes/lightspeed/imgs/logo-new-600.png",
          "width": 600,
          "height": 49
      },
      "name": "POPSUGAR",
      "sameAs": [
              "https://twitter.com/POPSUGAR",
                  "https://www.facebook.com/PopSugar",
                  "https://plus.google.com/+PopSugar",
                  "https://instagram.com/POPSUGAR",
                  "https://www.pinterest.com/POPSUGAR",
                  "https://www.youtube.com/popsugargirlsguide",
          "https://corp.popsugar.com"      ],
      "contactPoint": [
          {
              "@type": "ContactPoint",
              "telephone": "+1-415-391-7576",
              "contactType": "customer service"
          }
      ]
  }
</script>
```

Figure 3.10: A JSON-LD parsed Script



Πάτρικ Κούμπτ / Βιβλία

Learn SEO: An On-Page SEO Tutorial

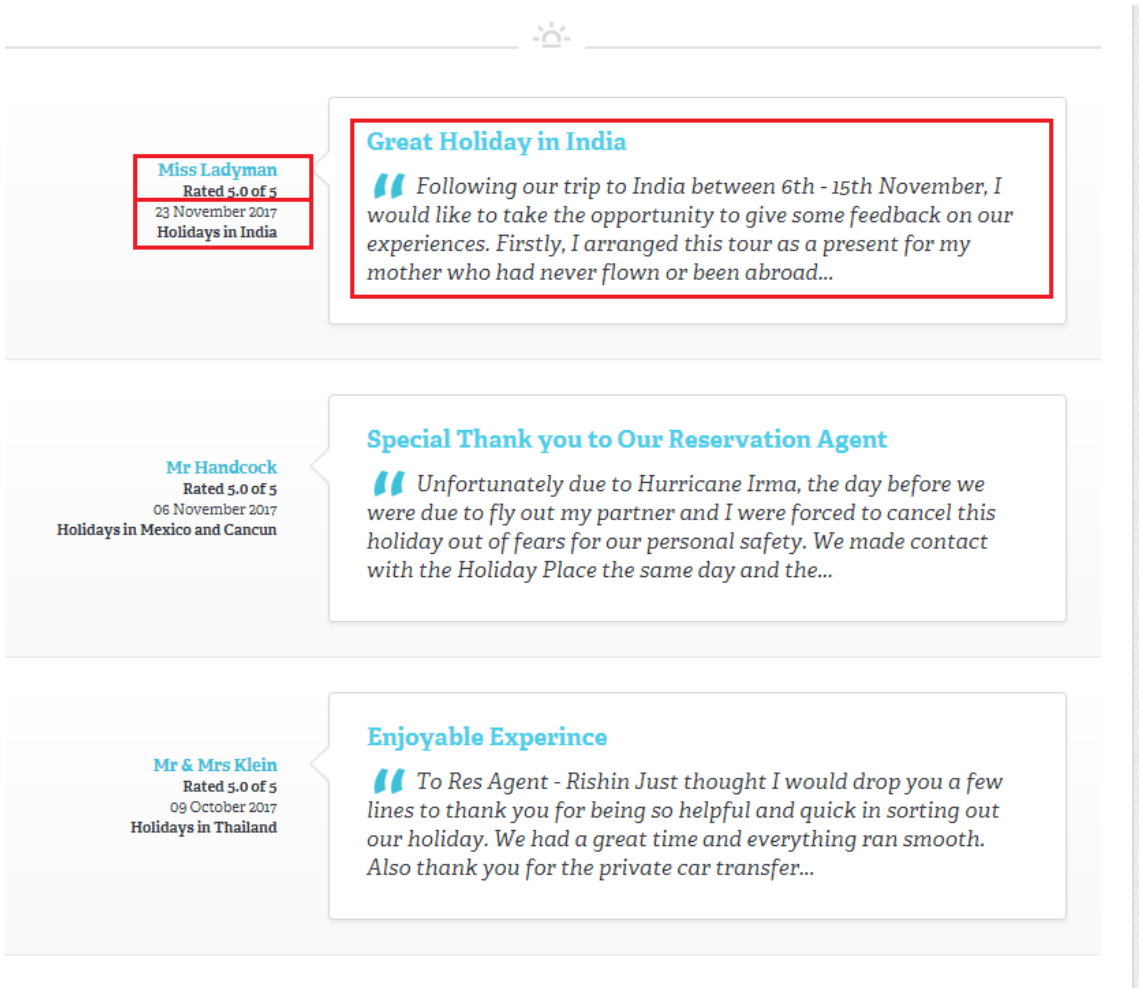
2015

Περισσότερα σχετικά με Learn SEO: An On-Page SEO Tutorial

Σχόλια

Amazon.com: Patrick Coombe: Books, Biography, Blog, Audiobooks ...
https://www.amazon.com/Patrick-Coombe/.../B014HX9... ▾ Μετάφραση αυτής της σελίδας
Are You an Author? Help us improve our Author Pages by updating your bibliography and submitting a new or current image and biography. › Learn more at Author Central · An On-Page SEO Tutorial. $19.99. Paperback. **Learn SEO: An On-Page SEO Tutorial**. $4.99. Kindle Edition ...

Figure 3.11: A Google search results that uses JSON-LD format.



Miss Ladyman
Rated 5.0 of 5
23 November 2017
Holidays in India

Great Holiday in India

Following our trip to India between 6th - 15th November, I would like to take the opportunity to give some feedback on our experiences. Firstly, I arranged this tour as a present for my mother who had never flown or been abroad...

Mr Handcock
Rated 5.0 of 5
06 November 2017
Holidays in Mexico and Cancun

Special Thank you to Our Reservation Agent

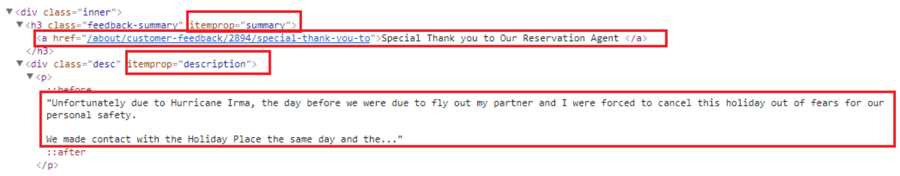
Unfortunately due to Hurricane Irma, the day before we were due to fly out my partner and I were forced to cancel this holiday out of fears for our personal safety. We made contact with the Holiday Place the same day and the...

Mr & Mrs Klein
Rated 5.0 of 5
09 October 2017
Holidays in Thailand

Enjoyable Experince

To Res Agent - Rishin Just thought I would drop you a few lines to thank you for being so helpful and quick in sorting out our holiday. We had a great time and everything ran smooth. Also thank you for the private car transfer...

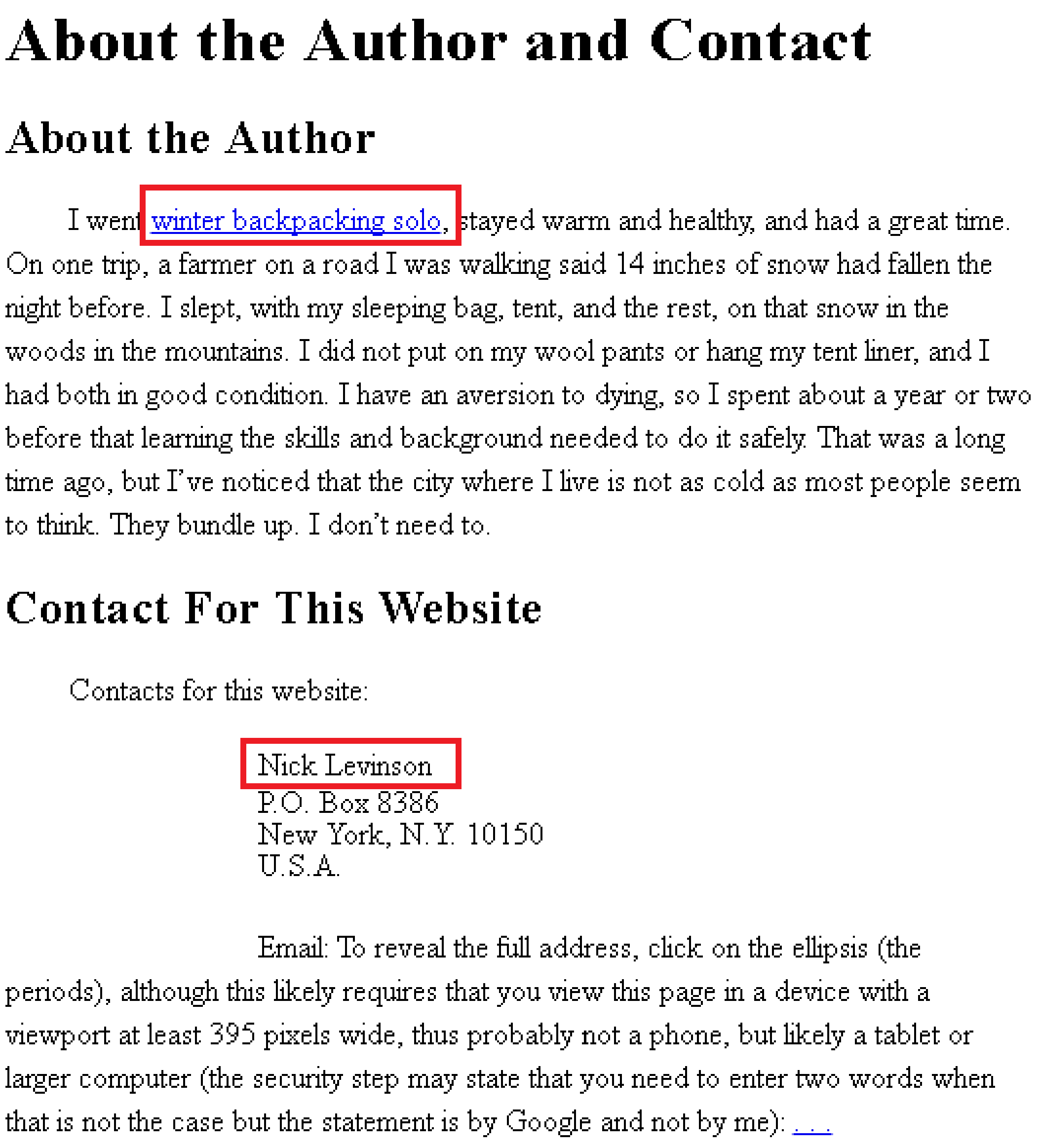
Figure 3.13: A feedback web page embedded with Microdata format.



```
▼<div class="inner">
 ▼<h3 class="feedback-summary" itemprop="summary">
    <a href="/about/customer-feedback/2894/special-thank-you-to">Special Thank you to Our Reservation Agent </a>
   </h3>
 ▼<div class="desc" itemprop="description">
   ▼<p>
      ::before
      "Unfortunately due to Hurricane Irma, the day before we were due to fly out my partner and I were forced to cancel this holiday out of fears for our
      personal safety.

      We made contact with the Holiday Place the same day and the..."
      ::after
    </p>
```

Figure 3.14: The microdata items (”itemprop”) from the web page.



About the Author and Contact

About the Author

I went winter backpacking solo, stayed warm and healthy, and had a great time. On one trip, a farmer on a road I was walking said 14 inches of snow had fallen the night before. I slept, with my sleeping bag, tent, and the rest, on that snow in the woods in the mountains. I did not put on my wool pants or hang my tent liner, and I had both in good condition. I have an aversion to dying, so I spent about a year or two before that learning the skills and background needed to do it safely. That was a long time ago, but I've noticed that the city where I live is not as cold as most people seem to think. They bundle up. I don't need to.

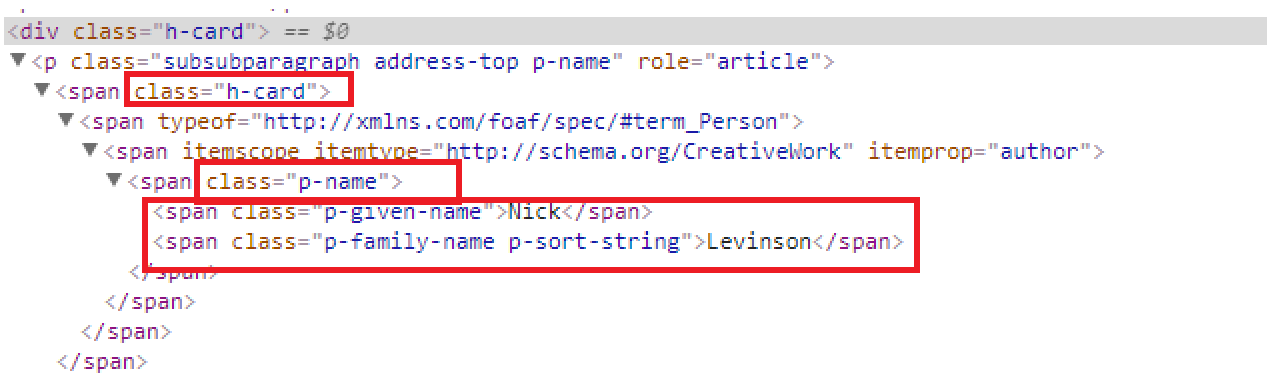
Contact For This Website

Contacts for this website:

Nick Levinson
P.O. Box 8386
New York, N.Y. 10150
U.S.A.

Email: To reveal the full address, click on the ellipsis (the periods), although this likely requires that you view this page in a device with a viewport at least 395 pixels wide, thus probably not a phone, but likely a tablet or larger computer (the security step may state that you need to enter two words when that is not the case but the statement is by Google and not by me): . . .

Figure 3.16: Web page using HCARD format.



```
<div class="h-card"> == $0
▼<p class="subsubparagraph address-top p-name" role="article">
  ▼<span class="h-card">
    ▼<span typeof="http://xmlns.com/foaf/spec/#term_Person">
      ▼<span itemscope itemtype="http://schema.org/CreativeWork" itemprop="author">
        ▼<span class="p-name">
            <span class="p-given-name">Nick</span>
            <span class="p-family-name p-sort-string">Levinson</span>
          </span>
        </span>
      </span>
    </span>
```

Figure 3.17: The HCARD format of the web page containing Microformats.

Chapter 4

Optimizations

The current chapter of this thesis is organized as follows: section 4.1 discusses on optimization index of SPARQL endpoints, section 4.2 presents a caching optimization of fetched datasets, section 4.3 introduces a caching optimization of returned service bindings, section 4.4 presents a parallel execution of `SERVICE` calls optimization, and finally an optimization based on reordering will be presented in section 4.5.

4.1 Index of Known SPARQL Endpoints

We have seen that, compared to the original `SERVICE` operator, the only additional cost is the time to run an `ASK` query (as we will see in §5, this cost is about 200 ms in average). To eliminate this cost, we can keep a small index with the URIs of known endpoints (like DBpedia's and Europeana's) as well as the URIs of endpoints that have been already checked. Thereby, if the `SERVICE` URI exists in the index, the query is directly forwarded to the endpoint, otherwise an `ASK` query is first submitted.

For example, consider the query of Figure 4.1. The query first retrieves Greek painters from the dereferenceable URI of the corresponding DBpedia category (lines 2-3), and then it queries Europeana' SPARQL endpoint for retrieving works of these painters (lines 4-6). However, if the number of painter URIs returned by the first `SERVICE` invocation is n, the query will call the remote endpoint n times (one for each painter URI), which in turn requires to run n `ASK` queries. Thus, in case we do not use the proposed index of known endpoints, the expected cost for running n `ASK` queries is about $n \times 200$ ms.

Experimental results are given in Section 5.2.1 .

4.2 Caching of Fetched Datasets

A SPARQL query may contain multiple `SERVICE` invocations against the same Web resource. Consider for example the query of Figure 3.4. In case the same

```
SELECT DISTINCT ?painter ?work  WHERE {
  SERVICE <http://dbpedia.org/resource/Category:Greek_painters> {
    ?painter <http://purl.org/dc/terms/subject> ?greekPainter }
  SERVICE <http://europeana.ontotext.com/sparql> {
    ?objectInfo <http://purl.org/dc/elements/1.1/creator> ?painter .
    ?objectInfo <http://www.openarchives.org/ore/terms/proxyFor> ?work } }
```

Figure 4.1: Example of a SPARQL query that calls the same remote SPARQL endpoint multiple times.

co-author exists in more than one publications, the corresponding RDF triples (of co-author's URI) will be redundantly fetched multiple times.

In such cases, fetching and loading repeatedly the same resource triples costs both in time, computer resources and traffic load. To avoid this, for a submitted query we can use a cache (usable only in the context of a submitted query) of datasets that have been already fetched. Thereby, in each new SERVICE invocation, we first check if the corresponding URI exists in the cache in order to avoid re-fetching its triples. The cache can be cleared after query execution. Of course, one could instead apply a caching policy that will keep the fetched resources in cache after query execution for serving future queries for a period of time and according to the available main memory, e.g. a combination of static and dynamic caching as it is used by web search engines [47].

Experimental results are given in Section 5.2.2 .

4.3 Request-scope Caching of SERVICE Bindings

There is the case of multiple SERVICE invocations against the same URI for the same graph pattern P. To avoid such redundant evaluations, for a submitted query we can cache the bindings resulted from each SERVICE-execution pair (URI, P). Then, for each new SERVICE invocation, if the corresponding (URI, P) pair exists in the cache, we abort its execution, return the cached bindings, and we continue to the next query stage. The cache is cleared after query execution.

For example, consider the query in Figure 4.2 The query first fetches all author URIs from the personal Web page of P. Fafalios (lines 2-3), and then, for each author URI, it directly queries the corresponding (dereferenceable) Web resource for retrieving and showing all of the author's papers. Since in the first SERVICE invocation we do not use the DISTINCT operator, the variable ?authorURI may be bound with the same URI more than one times (in case the same person is author in more that one papers). However, in this case the query executes the same graph pattern to the same fetched (and maybe cached, if we adopt the optimization described in §4.2) RDF triples multiple times. Thus, caching the bindings and re-using them can save time, especially in case the remote resource contains a very big number of triples and the graph pattern that must be executed over these triples contains costly operators (like `FILTER` and `OPTIONAL`).

Using this optimization we can get a modest speedup if the Group Pattern of the SERVICE call contains one of the following patterns: a) Filter pattern, b)Optional pattern or c)a sequence modifier such as distinct or order by. Experimental results are provided in Section 5.2.3 .

```
SELECT Distinct ?uris ?labels  WHERE {
 SERVICE <http://dbpedia.org/sparql> {
    ?uris a <http://dbpedia.org/ontology/Fish>;
          <http://dbpedia.org/property/name> ?name .}
 SERVICE <http://users.ics.forth.gr/~fafalios/data/100000.n3> {
   Select Distinct ?uris ?labels Where {
    ?uris ?p ?labels .
    FILTER (?p!= <http://www.w3.org/2000/01/rdf-schema#name>).
    FILTER (langMatches( lang(?labels), "en" )).
    FILTER( (contains(STR(?uris),"shark") &&
        contains(STR(?uris),"Great")) ||
        (regex(STR(?labels), "large", "i")) ) . }}
}
```

Figure 4.2: Example of a SPARQL query that may execute the same heavy load graph patterns to the same remote resource multiple times.

4.4 Parallel fetching of `Remote` resources

A SPARQL Query may consist of multiple SERVICE call invocations against different Web resources. In such cases we can fetch and cache these resources in a parallel fashion before executing the query. This optimization consists of two phases.

The starting phase occurs during the query processing stage where we retrieve all the IRIs SERVICE call operators. Then we can fetch the data from these IRIs, as long as they point towards HTTP resources (e.g., RDFa, microdata, JSON-LD, microformats) or RDF files (turtle, RDF/XML, N3, JSONLD) and not endpoints. The retrieval process is executed in a parallel for each individual IRI. In case the IRI directs towards endpoints the IRI of the endpoint is added to the list of known endpoints as described in 4.1.

For example consider the query depicted in figure 4.3. The query can retrieve data from the RDFa page that is pointed by the first SERVICE IRI, the ontology pointed from the second SERVICE IRI and add wikidata endpoint from the third SERVICE IRI to the list of known endpoints (4.1), in parallel.

In some instances the SERVICE operator is a variable. In such cases the SERVICE clause involving a variable can be executed as a series of separate invocations of SPARQL query services. The results of each invocation are combined using union. That said, the query engine needs to evaluate the variable results and retrieve the list of SERVICE IRIs. With this in mind, in the second phase, we can retrieve the additional data from the list of IRIs in parallel.

An example of a query is illustrated in figure 4.4. The query needs to first execute the first SERVICE invocation to yield bindings for the variable ?*uri* which will retrieve fish IRIs from dbpedia. We can now use this list of IRIs to fetch resources from these sources in parallel.

All things considered, we can benefit from this optimization if some of the remote sources that needs to be retrieved are quite large and therefore can delay the whole process when the data are fetched sequentially. However there is always the issue that the retrieved data may not have enough space to be stored in cache and consequently the query would fail.

```
SELECT Distinct ?s ?s2 ?s3 WHERE {
  SERVICE <http://users.ics.forth.gr/~fafalios> {
    ?s1 ?p1 ?o1  } (3.4s)
  SERVICE <http://www.ics.forth.gr/isl/ontology/MarineTLO/> {
    ?s2 ?p2 ?o2 . } (0.5s)
  SERVICE <https://www.wikidata.org/wiki/Q7132780> {
    ?s3 ?p3 ?o3  }}  (1.7s)
```

Figure 4.3: Example of a SPARQL query where we can fetch remote resources in parallel.

```
SELECT ?uri WHERE {
SERVICE  <http://dbpedia.org/sparql > {
   ?uri a dbo:Fish }
SERVICE ?uri { ?uri dbo:abstract ?abstr } }
```

Figure 4.4: Example of a SPARQL query with a variable ?*uri* as a SERVICE operator, where we can fetch remote resources in parallel.

4.5 Query reordering based on pattern restriction

In this section we will analyze our join ordering optimization using pattern restriction heuristics that is based on the syntax of SPARQL queries. We model the problem and provide 4 formulas based on heuristics.

4.5.1 Modeling

Let Q be a SPARQL query and let $S = (s_1, s_2, \ldots, s_n)$ be a *sequence* of n `SERVICE` patterns contained in Q. For a service pattern s_i, let g_i be its nested graph pattern and B_i be the list of bindings of Q *before* the execution of s_i. Our objective is to compute a reordering S' of S that minimizes its *execution cost.* Formally:

$$R^* = \underset{S'}{\operatorname{argmin}}\, cost(S') \tag{4.1}$$

In our case, the *execution cost* of a sequence of `SERVICE` patterns S' corresponds to its total execution time. However, the execution time of a `SERVICE` pattern $s_i \in S'$ highly depends on the query patterns that precede s_i, while the bindings produced by s_i affect the execution time of the succeeding `SERVICE` patterns. Considering the above, we can estimate $cost(S')$ as the weighted sum of the cost of each service pattern $s_i \in S'$ given B_i. Formally:

$$cost(S') = \sum_{i=1}^{n} (cost(s_i|Bi) \cdot w_i) \tag{4.2}$$

where $cost(s_i|Bi)$ expresses the cost of `SERVICE` pattern s_i given B_i (i.e., given the already-bound variables before executing s_i), and w_i is the weight of `SERVICE` pattern s_i which expresses the degree up to which it influences the execution time of the sequence S'. For example, one could define $w_i = \frac{n-i+1}{n}$, since its forthcoming `SERVICE` pattern is highly depended on the preceding `SERVICE` patterns in a query. In this case, for a sequence of four `SERVICE` patterns $S' = (s_1, s_2, s_3, s_4)$, the weights are: $w_1 = 1.0$ (since it influences the execution time of 3 more `SERVICE` patterns), $s_2 = 0.75$, $s_3 = 0.5$, and $s_4 = 0.25$.

Now, the cost of each `SERVICE` pattern s_i can be estimated based on the inverse selectivity (or unrestrictiveness) of its graph pattern g_i given B_i. Formally:

$$cost(s_i|Bi) = unrestrictiveness(g_i|Bi) \tag{4.3}$$

A `SERVICE` graph pattern that is very unrestrictive will return a big result set (big number of bindings), which in turn will increase the number of joins and the number of calls to succeeding `SERVICE` patterns, resulting in higher total execution time. Thus, our objective is to first execute the more restrictive `SERVICE` patterns that will probably return small result sets.

As proposed in [58] and [59] (for the case of triple patterns), the restrictiveness (or unrestrictiveness) of a graph pattern can be determined by the *number* and *type* of new (unbound) variables in the graph pattern. The most restrictive graph pattern can be considered the one containing the less unbound variables. Then, it is the one containing the less bounded variables. Subjects can be also considered more restrictive than objects, and objects more selective than predicates (usually there are more triples matching a predicate than a subject or an object, and more triples matching an object than a subject). Moreover, a triple that contains a literal instead of a URI in the object position is more restrictive since URIs can be also used as subjects and thus can return more data. Finally, the number and type of joins can also affect the restrictiveness of a graph pattern. Below, we define formulas for *unrestrictiveness* that consider the above factors.

4.5.2 Approaches

4.5.2.1 Variable Count (VC).

Our first unrestrictiveness measure considers only the number of graph pattern variables without considering whether they are bound or not. For a given graph pattern g_i, let $V(g_i)$ be the set of variables of g_i. The unrestrictiveness of g_i can be now simply defined as:

$$unrestrictiveness(g_i|Bi) = |V(g_i)| \tag{4.4}$$

With the above formula, the higher the number of variables in a graph pattern is, the higher is its unrestrictiveness score.

Consider the example in figure 4.5. The first SERVICE call, containing a triple pattern of 3 variables, is more likely to retrieve a higher number of results than the second one that contain only one. Therefore we can conclude that the second SERVICE call is more *restrictive* and thus more efficient to execute first.

```
SELECT ?s WHERE {
SERVICE  <http://lod.openlinksw.com/sparql/> {
   ?s ?p ?o }                 (#438M+)
SERVICE <http://dbpedia.org/sparql> {
        ?s a dbo:fish }        (#19392) }
```

Figure 4.5: Example of a VC optimization.

4.5.2.2 Unbound Variable Count (UVC).

Now, we can also consider the set of binding B_i before the execution of a SERVICE pattern s_i. Let first $V^u(g_i, B_i)$ be the set of new (unbound) variables of g_i given B_i. The unrestrictiveness of g_i can be now defined as:

$$unrestrictiveness(g_i|Bi) = |V^u(g_i, B_i)| \tag{4.5}$$

For example in figure 4.6, the first SERVICE call, that is retrieving all types of fishes from the order of Ostariophysi, is more likely to retrieve a small number of results than the rest. With that in mind executing the third SERVICE call second is the most beneficial since both of the variables of the triple pattern are bounded, compared to the second SERVICE call that still has the label variable unbound. Finally the fourth SERVICE call is the most costly, therefore it will be executed last.

```
SELECT Distinct ?uri ?label ?img  WHERE {
SERVICE <http://dbpedia.org/sparql> {
   ?uri dbr:order dbr:Ostariophysi }
SERVICE <http://users.ics.forth.gr/~yannakis/data/10000.n3> {        #10,000
  ?uri rdfs:label ?label  }
SERVICE <http://lod.openlinksw/sparql> {
    ?uri  <http://www.w3.org/2002/07/owl#differentFrom> ?uri. }
 SERVICE <https://query.wikidata.org/sparql>{
   ?uri2  foaf:depiction ?img; ?p ?o.}
}
```

Figure 4.6: Example of a UVC optimization.

4.5.2.3 Weighted Unbound Variable Count (WUVC).

The above formulas do not consider the type of the unbound variables in the graph pattern, i.e., whether they in the subject, predicate or object position in the triple pattern. For a graph pattern g_i and a set of bindings B_i, let $V_s^u(g_i, B_i)$, $V_p^u(g_i, B_i)$ and $V_o^u(g_i, B_i)$ be the set of subject, predicate and object unbound variables in g_i, respectively. Let also w_s, w_p and w_o be the weights for subject, predicate and object variables, respectively (where $w_s + w_p + w_o = 1.0$). We first define a *weighted variable score* which considers both the number and the type of unbound variables. The unrestrictiveness of g_i can be now defined as:

$$unrestrictiveness(g_i|Bi) = |V_s^u(g_i, B_i)| \cdot w_s + |V_p^u(g_i, B_i)| \cdot w_p + |V_o^u(g_i, B_i)| \cdot w_o \quad (4.6)$$

As proposed in [58] and [59], subjects can be considered more restrictive than objects, and objects more restrictive than predicates, which means that a subject variable may return more bindings than an object variable, and an object variable more bindings than a predicate variable. Thus, one can define values where $w_s > w_o > w_p$. From further studies [25] we can conclude that the possibility of a triple matching a predicate variable is 50% more than subject and 40% more than a object whereas an object is 10% more possible to be matched with a triple than a subject. Therefore we can define the weights based on this study as $w_s = 0.24$, $w_o = 0.27$, $w_p = 0.49$. Moreover, if a variable exists in more than one triple pattern position (e.g., both as subject or object), we consider it as being in the more restrictive position.

Consider the example in figure 4.7. The second `SERVICE` call is more likely to retrieve a higher number of results than the others. However the third one is the most restrictive according to the current weights. Therefore it is more efficient to execute the third `SERVICE` call first, then the first and finally the second one.

```
SELECT ?s ?o WHERE {
SERVICE <http://dbpedia.org/sparql> {                   #343
        ?s ?p dbo:Carcharhiniformes .} (1 s, 1 p)
SERVICE <https://query.wikidata.org/sparql> {           #33M+
        ?s owl:sameAs ?o .} (1 s, 1 o)
SERVICE <http://lod.openlinksw.com/sparql/> {           #128
   dbr:Shark ?p ?o .} (1 p, 1 o)
```

Figure 4.7: Example of a WUVC optimization.

4.5.2.4 Joins-aware Weighted Unbound Variable Count (JWUVC).

When a graph pattern contains joins, its restrictiveness is usually increased depending on the type of joins (e.g., chain, star or unusual join) [35] as well as by the number of joins. For a graph pattern g_i, let $J_s(g_i)$, $J_p(g_i)$ and $J_o(g_i)$ be the number of joined subjects, predicate and object in g_i, respectively. Let also j_t be the weight for the join type, respectively (where $0.5 < j_t < 1.0$), [35]. According to the study performed by Gallego [26], more than 90% of join types used in queries are star or chain queries. Thus in this work, we will only focus only on these two types as well as their combination. Bellow we define a formula which considers both the type, the number of unbound variables and the number of joins.

$$unrestrictiveness(g_i|Bi) = \frac{|V_s^u(g_i, B_i)| \cdot w_s}{1 + J_s \cdot j_t} + \frac{|V_p^u(g_i, B_i)| \cdot w_p}{1 + J_p \cdot j_t} + \frac{|V_o^u(g_i, B_i)| \cdot w_o}{1 + J_o \cdot j_t} \tag{4.7}$$

4.5.3 Discussion on Other Cases

If the query contains an OPTIONAL operator (or negation) then we only reorder the SERVICE calls above and bellow the OPTIONAL operator to preserve the correct results. Furthermore, when a Select query exist in a SERVICE pattern we only consider cost of the variables in the Select clause. Finally in FILTER cases we consider the filtered variables same as joined cases but the value of j_t for the JWUVC formula is higher.

4.5.4 Case of Variable in SERVICE Operator

In case of a variable SERVICE operator we take care to maintain its order in the query and therefore we reorder the upper and lower part independently. Using this method we make sure that the variable operator is bounded as well as that we retrieve the correct results. However SERVICE calls that don't contain this variable are free to be reordered without this restriction.

4.5.5 Case of frequent used properties

Frequent used properties such as rdfs:type or owl:sameAs are considered unrestrictive and are counted as variables in the `SERVICE` call.

4.5.6 Case of `SERVICE` pattern with Literal

In graph patterns the more bound a components is, the more selective the pattern is [61]. With this in mind we consider graph pattern with a higher number of literals more restrictive. This is especially useful when we consider join patterns in a `SERVICE` call to reduce the overall size of the join results. Similarly we conclude that literals are more selective than URIs.

4.5.7 Computation of Optimal Re-ordering

In this paper we aim at finding the best query plan, we search for the `SERVICE` call using the estimated costs using the previous heuristics and formulas. We do not generate all candidate query plans in advance because a trade-off is necessary when doing optimization, since the planning time needs to be counted as part of the execution time. Thus, this paper adopts a greedy algorithm starting at the `SERVICE` call with smaller cost and searching other linked `SERVICE` calls recursively. The purpose is to reduce the size of the data set to be queried as much as possible, as soon as possible. The algorithm balances between optimization time and accuracy of query planning in order not to spend too much resources on the task for optimizing the query. Hence, the algorithm may not find the most optimal solution in some cases due to the constraints of greedy algorithm.

4.5.8 Combination of Triple and SERVICE patterns

A basic query triple pattern has lower selectivity and higher cost than a named graph pattern. But when the named query triple pattern conforms to t(?, ?, ?), we will consider that it costs more than the basic query triple pattern, if the `SERVICE` pattern does not only contain query triple patterns like t(?, ?, ?). This would only affect Services that are bound by local variables. Since we want to reduce the total number of calls. A query executed with a specific named or `SERVICE` graph pattern has higher selectivity and costs less, for instance, graph foaf:bob { ... } costs less than `SERVICE` graph Service ¡URI¿ { ... } in a SPARQL query. Furthermore a Service with a variable operator is less restrictive than Services with a specific URI.

‘

Chapter 5

Experimental evaluation

We have seen that using `SPARQL-LD`, one can run queries which are more expressive than those supported by SPARQL 1.1. Nevertheless, here we evaluate the efficiency of the extended `SERVICE` operator for several querying scenarios, examining also the cost of each task of query execution.

We first present a sort synopsis of the datasets that were used for these experiments, in (§5.1). Then we evaluate the proposed optimizations i.e., index of known SPARQL endpoints, caching of fetched datasets, parallel fetching of remote resources and join ordering based on restriction estimation, in (§5.2). This allows us to investigate the efficiency for each of our optimizations.

The experiments were carried out using an ordinary computer with processor Intel Core i5 @ 2.8Ghz CPU, 8GB RAM and HDD 250 GByte running Windows 10 (64 bit). The implementation is in Java 1.8 on Apache ARQ 3.3 and ANY23 2.1.

5.1 Description of Datasets and Queries used for the experiments

In this section we will describe the characteristics of the queries that were used for the benchmarks as well as the properties of the datasets and endpoints that we run the experiments on. We run several set of queries, each containing multiple `SERVICE` calls or complex join patterns. For the first set of queries we increased the complexity of each `SERVICE` call while maintaining the number of `SERVICE` calls. For the second set of queries we increased the number of `SERVICE` calls per query while maintaining the same complexity.

5.1.1 Characteristics of endpoints

The endpoints used for this evaluation were, Dbpedia, Wikidata and Yago, as well as some rich JSON-LD pages and RDF files that were produced randomly, from DBpedia, ranging from $1,000$ to $1,000,000$ triples each.

Table 5.1: Datasets statistics.

Dataset	Yago	DBpedia	Wikidata
Triples	1,001,461,792	$1,436,545,545$	748,530,833
URIs	348,565,990	369,254,196	244,013,839
Literals	682,313,508	161,398,000	
Distinct subjects	331,807,591	31,391,000	142,278,154
Distinct predicates	106	2,819	1,874
Distinct objects	17,438,118	83,285,000	101,745,685

Table 5.2: Yago statistics on joins.

Dataset	Yago
$s \bowtie s$	165,937,025
$s \bowtie p$	9,396,314
$s \bowtie o$	48,560,142
$p \bowtie p$	65,992,300,043,771
$p \bowtie o$	0
$o \bowtie o$	30,079,265,139
Distinct $s \bowtie p$	82
Distinct $s \bowtie o$	2,169,728
Distinct $p \bowtie o$	0

DBpedia [4]: is a knowledge base containing content that has been converted from Wikipedia, that by the time of writing this thesis, the English version contained more than 20 million resources.

Wikidata [12]: is a free, collaborative, multilingual, secondary database, collecting structured data to provide support for Wikipedia, Wikimedia Commons [13], the other wikis of the Wikimedia movement, and to anyone in the world.

Yago [14]: is a huge semantic knowledge base, derived from Wikipedia, WordNet and GeoNames. Currently, YAGO has knowledge of more than 10 million entities (like persons, organizations, cities, etc.) and contains more than 120 million facts about these entities.

Statistics gathered from these endpoints are presented in fig. 5.1 and fig. 5.2.

Table 5.3: Effect of first optimization (*index of known endpoints*).

Query	Num of calls to indexed endpoints	Time without Opt.	Time with Opt.	Speedup
Q1	10	3.5 sec	1.8 sec	1.9×
Q2	10^2	27.2 sec	16.5 sec	1.6×
Q3	10^3	9.6 min	2.5 min	3.9×
Q4	10^4	44.8 min	24 min	1.9×
Q13	10	7.4 sec	3.2 sec	2.3×
Q14	10^2	36.1 sec	19.4 sec	1.9×
Q15	10^3	4.9 min	2.8 min	1.9×

5.2 Effect of Optimizations

We run experiments with and without the proposed optimizations (effectively comparing our system with the system in [23]). As regards the first optimization *(index of known endpoints)*, the expected speedup depends on the number of `SERVICE` calls to endpoints that exist in the index. As regards the second optimization *(caching of fetched datasets)*, the expected speedup depends on both the number of `SERVICE` calls to already-fetched resources and on the size (number of triples) of these resources. The queries used in this evaluation are available at `http://users.ics.forth.gr/~fafalios/sparql-ld/Eval.zip`. We run each query 7 times in a consecutive manner for each optimization technique and here we report the average values.

5.2.1 Index of Known SPARQL Endpoints evaluation

Regarding the first optimization, we run experiments for different number of calls to "known" remote endpoints (Dbpedia and Wikidata). Table 5.3 shows the speedup for each case. The speedup is calculated as the query execution time when the optimization is not applied divided by the optimized time. We notice that, using the proposed optimization method, the query execution time can be significantly improved (in our experiments, it is from 1.6 to 3.9 times faster).

5.2.2 Caching of fetched datasets evaluation

As regards the second optimization, we run experiments for different number of calls to already-fetched resources, for different number of triples in these resources and by increasing the number of duplicate IRIs in these resource. Table 5.4 shows the results. As expected, this optimization can highly improve the efficiency of query execution (in our experiments, it is from 1.2 to 24.6 times faster), while it also reduces the transfer of data between local server and remote sources. More information about the queries can be found in appendix 7.1 .

Table 5.4: Effect of second optimization (*caching of fetched datasets)*.

Query	Num of calls to cached datasets	Num of triples	Time without Opt.	Time with Opt.	Speedup
Q5	**16**	10^3	11.9 sec	1.4 sec	8.5×
Q6	**16**	10^4	72.9 sec	5.7 sec	12.7×
Q7	**16**	10^5	10.3 min	39.8 sec	15.5×
Q16	**16**	10^6	1.7 hours	5.2 min	19.2×
Q8	10	$\mathbf{10^2}$	11.8 sec	9.6 sec	1.2×
Q9	10^2	$\mathbf{10^2}$	35.9 sec	10.7 sec	3.4×
Q10	10^3	$\mathbf{10^2}$	4.8 min	11.6 sec	24.6×
Q17	$5 * 10^3$	$\mathbf{10^2}$	21.7 min	11.9 sec	115.2×
Q18	10^4	$\mathbf{10^2}$	39.2 min	13 sec	180.9×

Table 5.5: Effect of second optimization (*caching of fetched datasets) on duplicate URis.*

Query	Num of duplicate URIs	Time without Opt.	Time with Opt.	Speedup
Q19	44	62.7 sec	15.9 sec	3.9 ×
Q20	116	284 min	39.9 sec	4.2 ×

Table 5.6: Effect of third optimization (*parallel fetching of* `Remote` *resources).*

Query	Num of datasets	Num of triples per dataset	Time without Opt.	Time with Opt.	Speedup
Q8	10	$\mathbf{10^2}$	11.8 sec	10.6 sec	1.1×
Q9	10^2	$\mathbf{10^2}$	35.9 sec	29.9 sec	1.2×
Q10	10^3	$\mathbf{10^2}$	4.8 min	4.2 min	1.1×
Q36	10	$\mathbf{10^5}$	6.5 min	4.9 min	1.3×
Q37	10^2	$\mathbf{10^5}$	55.3 min	41.3 min	1.3×
Q38	$10 * 1$	$\mathbf{10^2}, \mathbf{10^5}$	18.1 sec	13.1 sec	1.4×
Q39	$10^2 * 10$	$\mathbf{10^2}, \mathbf{10^5}$	6.9 min	4.7 min	1.5×
Q40	$10^3 * 10^2$	$\mathbf{10^2}, \mathbf{10^5}$	62.2 min	34.8 min	1.8×

5.2.3 Parallel fetching of `Remote` resources evaluation

As regards the third optimization, we run experiments for different of small and large datasets, by increasing their number. We also used a mix of small and large datasets, for the third part. Table 5.6 shows the results. As expected, this optimization can highly improve the efficiency of query execution (in our experiments, it is from 1.1 to 1.8 times faster), especially when querying a mix of small and larger datasets.

Observing the results we can notice a decrease in speedup when we increase the number of small datasets. That is due to the parallel conflicts that happens when more than one dataset is retrieved at the exact same moment.

5.2.4 Query reordering evaluation

As regards the fourth optimization, we run experiments for 15 different types of query patterns. In particular we used increasingly more complex query patterns for the first 4 queries (e.g., queries Q21-Q24). Then we evaluated our system by increasing the number of `SERVICE` calls (e.g., queries Q26 - Q30) while maintaining the same complexity. And finally we evaluated different join patterns (e.g., star and chain join patterns) as well as a mix of the above (e.g., queries Q25, Q31 - Q35). Dotplot 5.1 shows the results. As expected, this optimization can achieve the optimal query plan more than 80%, improving the overall efficiency of query execution.

However, there are some results where our system didnt reach the optimal execution plan. That can be observed in 3 query patterns. The main reason for these results is due to the high complexity of joins where our planner was unable to reach the optimal plan. Moreover our system had difficulties when a FILTER case was used. Nonetheless when `SPARQL-LD` didn't achieve the best optimal execution plan it was still close to optimal, as we can observe from the results.

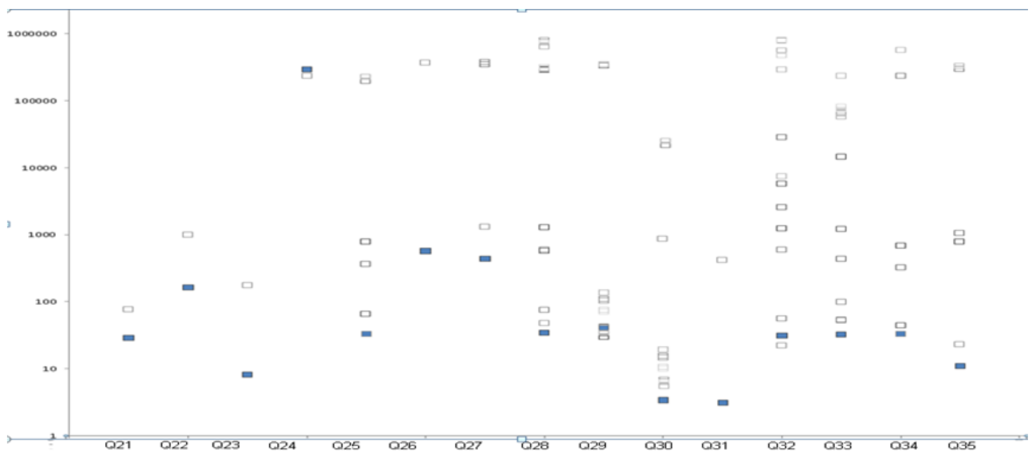


Figure 5.1: Reordering Evaluation.

5.2.4.1 Jena TDB Optimizer

Jena TDB optimizer [11] involves both static and dynamic optimizations, employing query reordering techniques or a statistics based strategy. These options are by default disabled and need to be selected explicitly.

In our system, we focused on static optimizations, i.e., transformations of the SPARQL algebra performed before query execution begins, without using any statistical data. Hence we will evaluate only the query reordering optimization. TDB uses a triple pattern reordering technique based on the number of variables in a triple pattern. This optimization decides the best order of triple patterns in a basic graph pattern after all of the variables have been bound.

With that said, by enabling TDB optimizer on top of our query reordering optimization, would improve the overall query execution process, especially in cases involving complex graph query patterns, e.g., queries Q22-Q24. First, by employing our query reordering technique we would reorder the `SERVICE` patterns of the query and then the algebra for each graph pattern would be rewritten to a better one by the TDB optimizer, further decreasing the query execution time. However in cases involving multiple number of `SERVICE` calls that contain single triple patterns, e.g., queries Q26-Q30, the TDB optimizer would fail to improve the query execution time.

5.3 Evaluation analysis

All of the above optimizations have satisfying results. More specific:

- For the first optimizations the experiments showed up to 3.9 times faster
- For the second optimizations the experiments showed up to 24.6 times faster
- For the third optimizations the experiments showed up to 1.8 times faster

- While for the fourth optimization, we reached close to 80% of optimal time

Specifically for the fourth optimizations and based on the results from DBpedia and Wikidata we can conclude that the best results came from the JWUVC formula. The statistics gathered actually confirm this results since the datasets contains a low count of unique predicates while maintaining a high unique count of objects per subject. We also observed that in the experiments where we increase the complexion of each `SERVICE` call the results are also in favor of WUVC formula but in this case also simple UVC returned good results. This is due to the increase of joins in the `SERVICE` pattern. However when our planner is evaluated over large star joins it fails to produce plans in a regular basis that are close to optimal.

Chapter 6

Conclusion and Future Work

In this work we propose a heuristics-based query planner for `SPARQL-LD` that uses a set of heuristics based on syntactic and structural characteristics of SPARQL graph patterns. To this end, we exploited query reordering techniques to produce plans that minimize the number of `SERVICE` calls to remote SPARQL endpoints, by selecting `SERVICE` call patterns that are most likely to have high selectivity first and thus reducing the intermediate results. In particular, we propose a set of formulas for deciding which `SERVICE` patterns of a SPARQL query are more selective, thus beneficial for the planner to evaluate first in order to reduce the intermediate results during query execution. These formulas are generic and can be used separately or complementary to each other as well as with other heuristic planners.

Specifically, in our work we propose the reduction of the query planning problem by modeling a SPARQL query as a sequence of `SERVICE` patterns, where each pattern is assigned an unrestrictiveness cost. Our objective is to first execute the more restrictive `SERVICE` pattern that will return the smallest result set. To the best of our knowledge no other work on query planning focuses on reordering `SERVICE` patterns.

The proposed optimizations are implemented on `SPARQL-LD`, a SPARQL 1.1 extension that allows to directly fetch and query RDF data from any HTTP Web source. Using the extended `SPARQL-LD` that was implemented in this thesis, one can exploit and combine in the same SPARQL query: i) data stored in the (local) repository, ii) data coming from online RDF or JSON-LD files, iii) data coming from dereferenceable URIs, iv) data that is dynamically created by Web services, v) data coming by querying other SPARQL endpoints and with the extension, vi)data embedded in Web pages as RDFa, Microdata, Microformats and JSON-LD. The functionality offered by `SPARQL-LD` motivates Web publishers to follow the Linked Data principles and offer their data in RDF without needing to set up and maintain a costly SPARQL endpoint. For instance, a museum can enrich its Web page with JSON-LD, or just put online an RDF dump, and thereby make its data directly accessible via SPARQL. In this thesis we focused on optimizations for SPARQL-LD, in particular on: (a) static methods for query reordering using selectivity estimation techniques on new (unbound) variables for increasing efficiency and decreasing the number of calls to remote sources, (b) methods for parallelizing RDF data retrieval for efficient data caching, (c) caching techniques for fetched datasets and (d) by using indexing to store the known SPARQL endpoints.

The experimental evaluation showed that they can highly improve the query execution time. Specifically, as regards (a), the experiments showed promising results, in particular we executed queries on popular SPARQL endpoints like DB-pedia, Wikidata and Yago. The results suggested that in most cases (90%) we came close to optimal results and improved effectively the query performance of original Jena system. However for queries that contain large star joins our heuristics cost model fails to produce near to optimal plans. Nevertheless our findings confirmed our underlying assumptions regarding the selectivity of subject-property-object components in a `SERVICE` pattern as well as the selectivity of the join patterns. As regards (b), the experiments showed a minor improvement when we fetch multiple resources that are small in size, up to 1.2× faster. However in case there were a mix of small and big fetched data the evaluation times produced better results, up to 2.1× faster. Both of these optimizations highly depend on the nature of data queried but in many cases the results where encouraging. Considering (c), the results showed a big improvement, up to 26× faster, since the reduction on transferring the data is between local server and remote sources is considerable. Regarding (d), the results showed a mild improvement, up to 3.9× faster.

6.1 Future Work

There are several directions worth further research, including:

- Machine learning re-ordering by using statistics from datasets or endpoints that were queried in the past.
- Extend `SPARQL-LD` to altogether bypass endpoints by transforming queries to endpoints to queries towards HTTP resources by using dereferencable URIs.
- Analyze more features from the SPARQL language such as the OPTIONAL and FILTER clause.
- Investigate the effects of applying our heuristics cost based query planning model in a distributed environment such as SPARK.

Chapter 7

Appendix

Here we list the queries that were used for our evaluations.
Queries Q1-Q4 and Q13-15 are used for the evaluation of the first optimization (i.e., index of known endpoints, subsection 5.2.1).
Queries Q5-Q10 and Q16-20 are used for the evaluation of the second optimization (i.e., caching of fetched datasets, subsection 5.2.2).
Queries Q8-Q10 and Q36-40 are used for the evaluation of the third optimization (i.e., parallel fetching of remote datasets, subsection 5.2.3).
Queries Q21-Q35 are used for the evaluation of the fourth optimization (i.e., query reordering, subsection 5.2.4).
The queries description is further analyzed in section 5.2.

7.1 Queries

```
SELECT Distinct ?uris ?labels  WHERE {
  SERVICE <http://users.ics.forth.gr/~fafalios/data/10.n3> {
    ?uris <http://www.w3.org/2000/01/rdf-schema#label> ?labels  }
  SERVICE <http://dbpedia.org/sparql> {
    ?uris a <http://dbpedia.org/ontology/Fish> }
}
```

Figure 7.1: **Q1** 10 calls to indexed endpoints

```
SELECT Distinct ?uris ?labels  WHERE {
  SERVICE <http://users.ics.forth.gr/~fafalios/data/100.n3> {
    ?uris <http://www.w3.org/2000/01/rdf-schema#label> ?labels  }
  SERVICE <http://dbpedia.org/sparql> {
    ?uris a <http://dbpedia.org/ontology/Fish> }
}
```

Figure 7.2: **Q2** 100 calls to indexed endpoints

```
SELECT Distinct ?uris ?labels  WHERE {
  SERVICE <http://users.ics.forth.gr/~fafalios/data/1000.n3> {
    ?uris <http://www.w3.org/2000/01/rdf-schema#label> ?labels  }
  SERVICE <http://dbpedia.org/sparql> {
     ?uris a <http://dbpedia.org/ontology/Fish> }
}
```

Figure 7.3: **Q3** 1,000 calls to indexed endpoints

```
SELECT Distinct ?uris ?labels  WHERE {
  SERVICE <http://users.ics.forth.gr/~fafalios/data/10000.n3> {
    ?uris <http://www.w3.org/2000/01/rdf-schema#label> ?labels  }
  SERVICE <http://dbpedia.org/sparql> {
    ?uris a <http://dbpedia.org/ontology/Fish> }
}
```

Figure 7.4: **Q4** 10,000 calls to indexed endpoints

```
SELECT Distinct ?fishuri ?name ?image WHERE {
  SERVICE <http://dbpedia.org/sparql> {
    ?fishuri <http://dbpedia.org/ontology/family>
                            <http://dbpedia.org/resource/Cyprinioidea> .
    ?fishuri <http://xmlns.com/foaf/0.1/depiction> ?image .
  SERVICE <http://users.ics.forth.gr/~fafalios/data/1000.n3> {
    ?fishuri <http://www.w3.org/2000/01/rdf-schema#label> ?name  }
}
```

Figure 7.5: **Q5** Constant number of calls to cached datasets – Different number of dataset's triples. 1,000 triples.

```
SELECT Distinct ?fishuri ?name ?image WHERE {
  SERVICE <http://dbpedia.org/sparql> {
    ?fishuri <http://dbpedia.org/ontology/family>
                               <http://dbpedia.org/resource/Cyprinioidea> .
    ?fishuri <http://xmlns.com/foaf/0.1/depiction> ?image .}
  SERVICE <http://users.ics.forth.gr/~fafalios/data/10000.n3> {
    ?fishuri <http://www.w3.org/2000/01/rdf-schema#label> ?name  }
}
```

Figure 7.6: **Q6** Constant number of calls to cached datasets – Different number of dataset's triples. 10,000 triples.

```
SELECT Distinct ?fishuri ?name ?image WHERE {
  SERVICE <http://dbpedia.org/sparql> {
    ?fishuri <http://dbpedia.org/ontology/family>
                              <http://dbpedia.org/resource/Cyprinioidea> .
    ?fishuri <http://xmlns.com/foaf/0.1/depiction> ?image .
  SERVICE <http://users.ics.forth.gr/~fafalios/data/100000.n3> {
    ?fishuri <http://www.w3.org/2000/01/rdf-schema#label> ?name  }
}
```

Figure 7.7: **Q7** Constant number of calls to cached datasets – Different number of dataset's triples. 100,000 triples.

```
SELECT DISTINCT ?personuri ?name ?comment WHERE {
  { SELECT ?personuri ?comment WHERE {
      SERVICE <http://dbpedia.org/sparql> {
        ?personuri a <http://dbpedia.org/ontology/Person> .
        ?personuri <http://www.w3.org/2000/01/rdf-schema#comment> ?comment
   } } LIMIT 10 }.
   SERVICE <http://users.ics.forth.gr/~fafalios/data/100.n3> {
      ?personuri <http://www.w3.org/2000/01/rdf-schema#label> ?name  }
}
```

Figure 7.8: **Q8** Constant number of calls to cached datasets – Different number of dataset's triples. 10 calls to cached datasets.

```
SELECT DISTINCT ?personuri ?name ?comment WHERE {
 { SELECT ?personuri ?comment WHERE {
     SERVICE <http://dbpedia.org/sparql> {
       ?personuri a <http://dbpedia.org/ontology/Person> .
       ?personuri <http://www.w3.org/2000/01/rdf-schema#comment> ?comment
   } } LIMIT 100}.
   SERVICE <http://users.ics.forth.gr/~fafalios/data/100.n3> {
      ?personuri <http://www.w3.org/2000/01/rdf-schema#label> ?name  }
}
```

Figure 7.9: **Q9** Constant number of calls to cached datasets – Different number of dataset's triples. 100 calls to cached datasetss.

```
SELECT DISTINCT ?personuri ?name ?comment WHERE {
 { SELECT ?personuri ?comment WHERE {
     SERVICE <http://dbpedia.org/sparql> {
        ?personuri a <http://dbpedia.org/ontology/Person> .
        ?personuri <http://www.w3.org/2000/01/rdf-schema#comment> ?comment
 } } LIMIT 1000}.
 SERVICE <http://users.ics.forth.gr/~fafalios/data/100.n3> {
    ?personuri <http://www.w3.org/2000/01/rdf-schema#label> ?name  }
}
```

Figure 7.10: **Q10** Constant number of calls to cached datasets – Different number of dataset's triples. 1,000 calls to cached datasetss.

```
SELECT DISTINCT ?authorURI ?paper WHERE {
   SERVICE <http://users.ics.forth.gr/~fafalios> {
     ?p <http://purl.org/dc/terms/creator> ?authorURI }
   SERVICE ?authorURI {
     ?paper <http://purl.org/dc/elements/1.1/creator> ?authorURI }
}
```

Figure 7.11: **Q11** Retrieve RDFa from fafalios personal page.

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT DISTINCT ?creator ?descr ?photo WHERE {
   SERVICE <http://europeana.ontotext.com/sparql> {
     ?work dc:subject dbr:Renaissance .
     ?work dc:creator ?creator FILTER(REGEX(STR(?creator),
                                       "^http://dbpedia"))}
   SERVICE ?creator {
    ?creator <http://purl.org/dc/terms/subject>
          dbr:Category:Mannerist_painters .
    ?creator <http://dbpedia.org/ontology/abstract> ?descr
    FILTER(lang(?descr)='en') .
    ?creator <http://xmlns.com/foaf/0.1/depiction> ?photo  }
}
```

Figure 7.12: **Q12** Retrieve renaissance artists and return their descriptions and photos.

```
 SELECT DISTINCT ?artistURI  (Count(?works) AS ?numOfWorks)  WHERE {
 {Select Distinct ?artistURI Where {
 SERVICE <http://dbpedia.org/sparql> {
   ?artistURI  <http://dbpedia.org/ontology/field> <http://dbpedia.org/resource/Pai
 SERVICE <http://europeana.ontotext.com/sparql> {
   ?objectInfo <http://purl.org/dc/elements/1.1/creator> ?artistURI  .
   ?objectInfo <http://www.openarchives.org/ore/terms/proxyFor> ?works.
   }
}Group By ?artistURI
```

Figure 7.13: **Q13** 10 calls to indexed endpoints.

```
SELECT DISTINCT ?artistURI  (Count(?works) AS ?numOfWorks)  WHERE {
 {Select Distinct ?artistURI Where {
 SERVICE <http://dbpedia.org/sparql> {
   ?artistURI  <http://dbpedia.org/ontology/field> <http://dbpedia.org/resource/Painting>}}
 SERVICE <http://europeana.ontotext.com/sparql> {
   ?objectInfo <http://purl.org/dc/elements/1.1/creator> ?artistURI  .
   ?objectInfo <http://www.openarchives.org/ore/terms/proxyFor> ?works.
   }
}Group By ?artistURI
```

Figure 7.14: **Q14** 100 calls to indexed endpoints.

```
SELECT DISTINCT ?artistURI  (Count(?works) AS ?numOfWorks)  WHERE {
 {Select Distinct ?artistURI Where {
 SERVICE <http://dbpedia.org/sparql> {
   ?artistURI  <http://dbpedia.org/ontology/field> <http://dbpedia.org/resource/Painting>}}
 SERVICE <http://europeana.ontotext.com/sparql> {
   ?objectInfo <http://purl.org/dc/elements/1.1/creator> ?artistURI  .
   ?objectInfo <http://www.openarchives.org/ore/terms/proxyFor> ?works.
   }
}Group By ?artistURI
```

Figure 7.15: **Q15** 1,000 calls to indexed endpoints.

```
SELECT Distinct ?fishuri ?name ?image WHERE {
  SERVICE <http://dbpedia.org/sparql> {
    ?fishuri <http://dbpedia.org/ontology/family>
                             <http://dbpedia.org/resource/Cyprinioidea> .
    ?fishuri <http://xmlns.com/foaf/0.1/depiction> ?image .
  SERVICE <http://users.ics.forth.gr/~fafalios/data/1000000.n3> {
    ?fishuri <http://www.w3.org/2000/01/rdf-schema#label> ?name  }
}
```

Figure 7.16: **Q16** 100 triples .

```
SELECT DISTINCT ?personuri ?name ?comment WHERE {
 { SELECT ?personuri ?comment WHERE {
    SERVICE <http://dbpedia.org/sparql> {
       ?personuri a <http://dbpedia.org/ontology/Person> .
       ?personuri <http://www.w3.org/2000/01/rdf-schema#comment> ?comment
 } } LIMIT 5000}.
 SERVICE <http://users.ics.forth.gr/~fafalios/data/100.n3> {
   ?personuri <http://www.w3.org/2000/01/rdf-schema#label> ?name  }
}
```

Figure 7.17: **Q17** 5,000 calls to cached datasets .

```
SELECT DISTINCT ?personuri ?name ?comment WHERE {
 { SELECT ?personuri ?comment WHERE {
    SERVICE <http://dbpedia.org/sparql> {
       ?personuri a <http://dbpedia.org/ontology/Person> .
       ?personuri <http://www.w3.org/2000/01/rdf-schema#comment> ?comment
 } } LIMIT 10000}.
 SERVICE <http://users.ics.forth.gr/~fafalios/data/100.n3> {
   ?personuri <http://www.w3.org/2000/01/rdf-schema#label> ?name  }
}
```

Figure 7.18: **Q18** 10,000 calls to cached datasets

```
SELECT DISTINCT ?authorName ?paper WHERE {
 SERVICE <http://users.ics.forth.gr/~fafalios/> {
   ?p <http://purl.org/dc/terms/creator> ?author
   FILTER(?author !=
   <http://dblp.l3s.de/d2r/resource/authors/Pavlos_Fafalios>) }
 SERVICE ?author {
   ?author <http://xmlns.com/foaf/0.1/name> ?authorName .
   ?paper <http://purl.org/dc/elements/1.1/creator> ?author }
 }
```

Figure 7.19: **Q19** Evaluating duplicate URIs.44 duplicate URIs

```
SELECT DISTINCT ?creator ?descr ?photo WHERE {
 SERVICE <http://europeana.ontotext.com/sparql> {
   ?work dc:subject dbr:Renaissance .
   ?work dc:creator ?creator FILTER(REGEX(STR(?creator),
         "^http://dbpedia"))}
  SERVICE ?creator {
   ?creator <http://purl.org/dc/terms/subject>
         dbr:Category:Mannerist_painters .
   ?creator <http://dbpedia.org/ontology/abstract> ?descr
                     FILTER(lang(?descr)='en') .
   ?creator <http://xmlns.com/foaf/0.1/depiction> ?photo  }
}
```

Figure 7.20: **Q20** Evaluating duplicate URIs. 116 duplicate URIs

```
SELECT Distinct ?battle  where {
SERVICE <https://query.wikidata.org/sparql> {
   ?battle ?p "Battle of Gettysburg" .  }
SERVICE <http://dbpedia.org/sparql> {
   res:Battle_of_Gettysburg owl:sameAs ?battle.  }
}
```

Figure 7.21: **Q21** A simple query to find the IRI of the battle of Gettysburg in wikidata endpoint. This example focus on the number of variables in a Service call. .

```
SELECT Distinct ?battle ?abstr ?place  where {
SERVICE <https://query.wikidata.org/sparql> {
   ?battle ?p "Battle of Gettysburg" ;
         wdt:P625 ?place.}
SERVICE <http://dbpedia.org/sparql> {
   res:Battle_of_Gettysburg owl:sameAs ?battle;
           dbo:abstract ?abstr. }
}
```

Figure 7.22: **Q22** A simple query to find the IRI of the battle of Gettysburg in wikidata endpoint. This example focus on the number of variables in a Service call with a more complex pattern.

```
SELECT Distinct ?battle ?abstr ?place  where {
SERVICE <https://query.wikidata.org/sparql> {
    ?battle ?p "Battle of Gettysburg" ;
         wdt:P625 ?place. }
SERVICE <http://dbpedia.org/sparql> {
   res:Battle_of_Gettysburg owl:sameAs ?battle;
           dbo:abstract ?abstr. FILTER (lang(?abstr) = "en" ) }
}
```

Figure 7.23: **Q23** A simple query to find the IRI of the battle of Gettysburg in wikidata endpoint.

```
Select  ?item ?name WHERE {
           SERVICE <http://dbpedia.org/sparql/> {
               ?item a dbo:Artist;
                  dbp:name ?name;
                  dbo:birthDate ?bd;
                  owl:sameAs ?item2.
                  Filter (?bd < "1900-01-01"^^xsd:dateTime ) }
SERVICE <https://query.wikidata.org/sparql> {
               ?item2 ?o wd:Q483501;
                   wdt:P569 ?db2;
                       rdfs:label ?name2 .
                       Filter (lang(?name2)="en") .
                 Filter (?db2 < "1900-01-01"^^xsd:dateTime )}
}
```

Figure 7.24: **Q24** A query where we get all artist names birthdate as well as urls to other endpoints and compare them from the data in wikidata to retrieve only the English labels.

```
Select  ?item ?name WHERE {
   SERVICE <http://dbpedia.org/sparql/> {
               ?item a dbo:Artist;
                  dbp:name ?name;
                  dbo:birthDate ?bd;
                  owl:sameAs ?item2.
                  Filter (?bd < "1900-01-01"^^xsd:dateTime ) }
   SERVICE <https://query.wikidata.org/sparql> {
               ?item2 ?o wd:Q483501;
                   wdt:P569 ?db2;
                       rdfs:label ?name2 .
                       Filter (lang(?name2)="en") .
                 Filter (?db2 < "1900-01-01"^^xsd:dateTime )}
   SERVICE <http://users.ics.forth.gr/~yannakis/files/1000.n3>{
               ?item a dbo:Artist.
}
 }
```

Figure 7.25: **Q25** A simple query where we load the graph for a given OCLC record URI, extracting the predicate for the OCLC Number and then querying Wikidata's SPARQL endpoint based on that number.

```
select Distinct ?authors ?works  where {
       Service <http://dbpedia.org/sparql/>{
               ?authors dbo:notableWork ?works.}
       Service <http://dbpedia.org/sparql/>{
               ?authors dbo:influencedBy dbr:Jules_Verne.}
}
```

Figure 7.26: **Q26** A query to find notable works influenced by Jules Verne.

```
A query to find the notable works influenced by Jules Verne that are also Science f
 select Distinct ?authors ?works  where {
        Service <http://dbpedia.org/sparql/>{
                ?authors dbo:notableWork ?works.}
        Service <http://dbpedia.org/sparql/>{
                ?authors dbo:influencedBy dbr:Jules_Verne.}
        Service <http://dbpedia.org/sparql/>{
                ?authors ?p2 dbr:Science_fiction.}
}
```

Figure 7.27: **Q27** A query to find notable works influenced by Jules Verne.

```
 select Distinct ?authors ?p ?values  where {
        Service <http://dbpedia.org/sparql/>{
                ?authors dbo:notableWork ?works.}
        Service <http://dbpedia.org/sparql/>{
                ?authors dbo:influencedBy dbr:Jules_Verne.}
        Service <http://dbpedia.org/sparql/>{
                ?authors ?p ?values.}
        Service <http://dbpedia.org/sparql/>{
                ?authors ?p2 dbr:Science_fiction.}
}
```

Figure 7.28: **Q28** A query to retrieve all English information from the notable works influenced by Jules Verne, that is also Science fiction.

```
select Distinct ?authors ?p ?values  where {
       Service <http://dbpedia.org/sparql/>{
               ?authors dbo:notableWork ?works.}
       Service <http://dbpedia.org/sparql/>{
               ?authors dbo:influencedBy dbr:Jules_Verne.}
       Service <http://dbpedia.org/sparql/>{
               ?authors ?p ?values.}
       Service <http://dbpedia.org/sparql/>{
               ?authors ?p2 dbr:Science_fiction.}
       Service <http://dbpedia.org/sparql/>{
               ?authors owl:sameAs ?authors2. }
}
```

Figure 7.29: **Q29** A query to retrieve all English information from the notable works influenced by Jules Verne, which is also Science fiction. Furthermore we want all related writers that have a URI in Wikidata.

```
select Distinct ?authors ?p ?values  where {
       Service <http://dbpedia.org/sparql/>{
               ?authors dbo:notableWork ?works.}
       Service <http://dbpedia.org/sparql/>{
               ?authors dbo:influencedBy dbr:Jules_Verne.}
       Service <http://dbpedia.org/sparql/>{
               ?authors ?p ?values.}
       Service <http://dbpedia.org/sparql/>{
               ?authors ?p2 dbr:Science_fiction.
       Service <http://dbpedia.org/sparql/>{
               ?authors owl:sameAs ?authors2. }
       Service <https://query.wikidata.org/sparql>{
               ?authors2 wdt:P27 wd:Q34266. }
}
```

Figure 7.30: **Q30** A query to retrieve all English information from the notable works influenced by Jules Verne, which is also Science fiction. Furthermore we want all related writers that have a URI in Wikidata and are Russian.

```
SELECT ?work ?name WHERE {
       SERVICE <http://lod.openlinksw.com/sparql/> {
               ?uri dct:subject ?oclcNumber.
               ?uri2 dct:creator ?oclcNumber.  }
       SERVICE <https://query.wikidata.org/sparql> {
               ?work wdt:P243 ?oclcNumber.
               ?work wdt:P50 ?name.  }
}
```

Figure 7.31: **Q31** A simple query where we load the graph for a given OCLC record URI, extracting the predicate for the OCLC Number and then querying Wikidata's SPARQL endpoint based on that number.

```
SELECT Distinct ?uri ?label ?img  WHERE {
  SERVICE <http://users.ics.forth.gr/~yannakis/data/1000.n3> {
     ?uri rdfs:label ?label.  }
  SERVICE <http://dbpedia.org/sparql> {
     ?uri a dbo:Fish; dbo:family dbr:Cyprinid. }
  SERVICE <http://lod.openlinksw/sparql> {
     ?uri2  owl:sameAs ?sameuri.
     ?sameuri dbo:order ?uri.
     ?uri dct:subject dbc:Fish_of_Europe.}
   SERVICE <https://query.wikidata.org/sparql>{
     ?uri2  foaf:depiction ?img; ?p ?o.}
}
```

Figure 7.32: **Q32** Get all labels from local file. But return only those that are type Fish, exist in openlinksw but exist in Europe and have an image depiction. Also return the URL and the image depiction.

```
SELECT Distinct ?uri ?class  ?descr  WHERE {
 SERVICE <http://lod.openlinksw/sparql> {
    <http://dbpedia.org/ontology/weight> a ?class.
    ?property a ?class.
    ?property2 a ?class. Filter(?property!=?property2). }
 SERVICE <http://lod.openlinksw/sparql> {
    ?property  rdfs:label ?label.
    ?property  rdfs:range ?range.
    ?property <http://www.w3.org/2007/05/powder-s#describedby> ?file}
  SERVICE ?file {
    ?uri  <http://open.vocab.org/terms/describes> ?descr.}
}
```

Figure 7.33: **Q33** Get all functional properties and then retrieve their range label and the RDF file that defines them. Then for each file return the descriptions.

```
SELECT ?journal ?journal_info WHERE {
   SERVICE <http://dbpedia.org/sparql> {
       ?dbp_journal a dbo:AcademicJournal.
       ?dbp_journal dct:subject dbc:Toxicology_journals.
       ?dbp_journal owl:sameAs ?journal.
       ?dbp_journal rdfs:label ?label.}
   SERVICE <https://query.wikidata.org/sparql> {
       ?journal wdt:P31 wd:Q5633421.
       ?journal wdt:P571 "1978-01-01"^^xsd:dateTime.
       ?journal wdt:P495 wd:Q145.}
   SERVICE <http://lod.openlinksw.com/sparql/>
       ?journal dc:title ?journal_info.}
}
```

Figure 7.34: **Q34** Get all toxological journals from dbpedia. Then retrieves only journals published form U.K. in 1978, found in wikidata. And finally returns the title from yago.

```
SELECT Distinct ?uri ?page ?img  WHERE {
  SERVICE <http://dbpedia.org/sparql> {
     ?uri a dbo:Fish.
     ?page <http://xmlns.com/foaf/0.1/primaryTopic> ?uri. }
  SERVICE <http://lod.openlinksw/sparql> {
      ?uri2 owl:sameAs ?uri.
      ?uri  wdt:P171 \Thunnini";
                     dbo:abstract ?abstr.}
   SERVICE <https://query.wikidata.org/sparql>{
     ?uri  foaf:depiction ?img; ?p ?o.}
}
```

Figure 7.35: **Q35** Get all fishes and their wiki pages. But return only those that exist in openlinksw, belong to genre Thunnini and have an image depiction. Also return the URL of the wiki page and the image depiction.

```
SELECT DISTINCT ?fish WHERE {
   SERVICE <http://dbpedia.org/sparql> {
            { SELECT ?personuri ?comment WHERE {
                ?personuri a <http://dbpedia.org/ontology/Person> .
                } LIMIT 10}
   }
   SERVICE <http://users.ics.forth.gr/~fafalios/data/100000.n3> {
      ?personuri <http://www.w3.org/2000/01/rdf-schema#label> ?name  }
}
```

Figure 7.36: **Q36** Constant number of calls to cached datasets – Different number of dataset's triples. 10 calls to cached datasets.

```
SELECT DISTINCT ?personuri ?name ?comment WHERE {
 SERVICE <http://dbpedia.org/sparql> {
           { SELECT ?personuri ?comment WHERE {
               ?personuri a <http://dbpedia.org/ontology/Person> .
               } LIMIT 100}
  }
  SERVICE <http://users.ics.forth.gr/~fafalios/data/100000.n3> {
     ?personuri <http://www.w3.org/2000/01/rdf-schema#label> ?name  }
}
```

Figure 7.37: **Q37** Constant number of calls to datasets – Different number of dataset's triples. 100 calls to datasetss.

```
SELECT DISTINCT ?personuri ?name ?comment WHERE {
  SERVICE <http://users.ics.forth.gr/~yannakis/files/100000.n3> {
           { SELECT ?uri WHERE {
               ?uri rdfs:label ?label .
               } LIMIT 10}
  }
  SERVICE <http://users.ics.forth.gr/~yannakis/files/1000.n3> {
     ?uri <http://www.w3.org/2000/01/rdf-schema#label> ?name  }
}
```

Figure 7.38: **Q38** Constant number of calls to datasets – Different number of dataset's triples. 100 calls to datasetss. With 10 calls to smaller dataset.

```
SELECT DISTINCT ?personuri ?name ?comment WHERE {
  SERVICE <http://users.ics.forth.gr/~yannakis/files/100.n3> {
            { SELECT ?personuri WHERE {
                ?personuri rdfs:label ?label .
                } LIMIT 10}
   }
  SERVICE <http://users.ics.forth.gr/~yannakis/files/100000.n3> {
            { SELECT ?personuri WHERE {
                ?personuri rdfs:label ?label2 .
                } LIMIT 100}
   }
   SERVICE <http://users.ics.forth.gr/~yannakis/files/1000.n3> {
      ?personuri <http://www.w3.org/2000/01/rdf-schema#label> ?name  }
}
```

Figure 7.39: **Q39** Constant number of calls to datasets – Different number of dataset's triples. 100 calls to datasetss. With 100 calls to smaller dataset.

```
SELECT DISTINCT ?personuri ?name ?comment WHERE {
  SERVICE <http://users.ics.forth.gr/~yannakis/files/100.n3> {
            { SELECT ?personuri WHERE {
                ?personuri rdfs:label ?label .
                } LIMIT 100}
   }
  SERVICE <http://users.ics.forth.gr/~yannakis/files/100000.n3> {
            { SELECT ?personuri WHERE {
                ?personuri rdfs:label ?label2 .
                } LIMIT 1000}
   }
   SERVICE <http://users.ics.forth.gr/~yannakis/files/1000.n3> {
      ?personuri <http://www.w3.org/2000/01/rdf-schema#label> ?name  }
}
```

Figure 7.40: **Q40** Constant number of calls to cached datasets – Different number of dataset's triples. 1,000 calls to cached datasetss. With 1000 calls to smaller dataset.

Bibliography

[1] A JSON-based Serialization for Linked Data. `http://www.w3.org/TR/json-ld/`.

[2] Apache Anything To Triples (ANY23). `https://any23.apache.org/`.

[3] Apache Jena. `http://jena.apache.org/`.

[4] DBpedia Endpoint. `http://dbpedia.org/sparql`.

[5] HTML Microdata. `https://www.w3.org/TR/microdata/`.

[6] HTML Microformats. `http://microformats.org/`.

[7] RDFa Core 1.1. `http://www.w3.org/TR/2015/REC-rdfa-core-20150317/`.

[8] SPARQL 1.1 Query Language (W3C). `http://www.w3.org/TR/sparql11-query/`.

[9] SPARQL Federat. Query. `http://www.w3.org/TR/sparql11-federated-query/`.

[10] SPARQL-LD. `https://github.com/fafalios/sparql-ld`.

[11] TDB Optimizer. `https://jena.apache.org/documentation/tdb/optimizer.html`.

[12] Wikidata Endpoint. `https://query.wikidata.org/`.

[13] Wikimedia Foundation. `https://wikimediafoundation.org/wiki/Home`.

[14] YAGO dataset. `https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/#c10444`.

[15] Renzo Angles and Claudio Gutierrez. The expressive power of sparql. In *International Semantic Web Conference*, pages 114–129. Springer, 2008.

[16] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[17] Abraham Bernstein, Christoph Kiefer, and Markus Stocker. *OptARQ: A SPARQL optimization approach based on triple pattern selectivity estimation.* University, 2007.

[18] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 5(3):1–22, 2009.

[19] Carlos Buil-Aranda, Marcelo Arenas, Oscar Corcho, and Axel Polleres. Federating queries in sparql 1.1: Syntax, semantics and evaluation. *Web Semantics: Science, Services and Agents on the World Wide Web*, 18(1), 2013.

[20] Carlos Buil-Aranda, Aidan Hogan, Jürgen Umbrich, and Pierre-Yves Vandenbussche. Sparql web-querying infrastructure: Ready for action? In *ISWC 2013*. Springer, 2013.

[21] Drashty R Dadhaniya and Ashwin Makwana. Survey paper for different sparql query optimization techniques. *Multi-disciplinary Journal of Scientific Research & Education*, 2(8), 2016.

[22] P. Fafalios, M. Baritakis, and Y. Tzitzikas. Exploiting linked data for open and configurable named entity extraction. *International Journal on Artificial Intelligence Tools*, 24(02), 2015.

[23] P. Fafalios and Y. Tzitzikas. SPARQL-LD: A SPARQL Extension for Fetching and Querying Linked Data. In *The Semantic Web–ISWC 2015 (Posters & Demonstrations Track)*, Bethlehem, Pennsylvania, USA, 2015.

[24] Pavlos Fafalios, Thanos Yannakis, and Yannis Tzitzikas. Querying the web of data with sparql-ld.

[25] Michael Färber, Frederic Bartscherer, Carsten Menne, and Achim Rettinger. Linked data quality of dbpedia, freebase, opencyc, wikidata, and yago. *Semantic Web*, (Preprint):1–53, 2016.

[26] Mario Arias Gallego, Javier D Fernández, Miguel A Martínez-Prieto, and Pablo de la Fuente. An empirical study of real-world sparql queries. In *USEWOD workshop*, 2011.

[27] Jinghua Groppe, Sven Groppe, Sebastian Ebers, and Volker Linnemann. Efficient processing of sparql joins in memory by dynamically restricting triple patterns. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1231–1238. ACM, 2009.

[28] Andreas Harth and Stefan Decker. Optimized index structures for querying rdf from the web. In *Web Congress, 2005. LA-WEB 2005. Third Latin American*, pages 10–pp. IEEE, 2005.

[29] Olaf Hartig. Sparql for a web of linked data: semantics and computability. In *9th ISWC*. Springer-Verlag, 2012.

[30] Olaf Hartig, Christian Bizer, and Johann-Christoph Freytag. *Executing SPARQL queries over the web of linked data*. Springer, 2009.

[31] Olaf Hartig and Ralf Heese. The sparql query graph model for query optimization. In *European Semantic Web Conference*, pages 564–578. Springer, 2007.

[32] Olaf Hartig and Jorge Pérez. Ldql: A query language for the web of linked data. In *The Semantic Web-ISWC 2015*, pages 73–91. Springer, 2015.

[33] Bernhard Haslhofer, Elaheh Momeni Roochi, Bernhard Schandl, and Stefan Zander. Europeana rdf store report. 2011.

[34] A. Hogan, A. Harth, J. Umbrich, S. Kinsella, A. Polleres, and S. Decker. Searching and Browsing Linked Data with SWSE: The Semantic Web Search Engine. *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(4), 2011.

[35] Hai Huang and Chengfei Liu. Estimating selectivity for joined rdf triple patterns. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1435–1444. ACM, 2011.

[36] Antoine Isaac and Bernhard Haslhofer. Europeana linked open data–data. europeana. eu. *Semantic Web*, 4(3):291–297, 2013.

[37] Graham Klyne and Jeremy J Carroll. Resource description framework (RDF): Concepts and abstract syntax. 2006.

[38] Krys J Kochut and Maciej Janik. Sparqler: Extended sparql for semantic association discovery. In *European Semantic Web Conference*, pages 145–159. Springer, 2007.

[39] Andreas Langegger, Wolfram Wöß, and Martin Blöchl. A semantic web middleware for virtual data integration on the web. In *5th ESWC*. Springer-Verlag, 2008.

[40] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 2014.

[41] Chang Liu, Haofen Wang, Yong Yu, and Linhao Xu. Towards efficient sparql query processing on rdf data. *Tsinghua Science & Technology*, 15(6):613–622, 2010.

[42] Daniel Miranker, Rodolfo Depena, Hyunjoon Jung, Juan Sequeda, and Carlos Reyna. Diamond: A sparql query engine, for linked data based on the rete match. *AImWD 2012*, 2012.

[43] Gabriela Montoya, Maria-Esther Vidal, and Maribel Acosta. A heuristic-based approach for planning federated sparql queries. *COLD*, 905, 2012.

[44] Hannes Mühleisen and Christian Bizer. Web data commons-extracting structured data from two large web corpora. *LDOW*, 937:133–145, 2012.

[45] Thomas Neumann and Gerhard Weikum. Rdf-3x: a risc-style engine for rdf. *Proceedings of the VLDB Endowment*, 1(1):647–659, 2008.

[46] Eyal Oren, Renaud Delbru, Michele Catasta, Richard Cyganiak, Holger Stenzhorn, and Giovanni Tummarello. Sindice.com: a Document-Oriented Lookup Index for Open Linked Data. *Int. J. Metadata Semant. Ontologies*, 3(1):37–52, 2008.

[47] Myron Papadakis and Yannis Tzitzikas. Answering keyword queries through cached subqueries in best match retrieval models. *Journal of Intelligent Information Systems*, 44(1):67–106, 2015.

[48] Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. In *International semantic web conference*, volume 4273, pages 30–43. Springer, 2006.

[49] Gregory Piatetsky-Shapiro and Charles Connell. Accurate estimation of the number of tuples satisfying a condition. *ACM Sigmod Record*, 14(2):256–276, 1984.

[50] Reinhard Pichler and Sebastian Skritek. Containment and equivalence of well-designed sparql. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 39–50. ACM, 2014.

[51] Eric Prud, Andy Seaborne, et al. Sparql query language for rdf. 2006.

[52] Bastian Quilitz and Ulf Leser. Querying distributed rdf data sources with sparql. In *5th ESWC*. Springer, 2008.

[53] Max Schmachtenberg, Christian Bizer, and Heiko Paulheim. Adoption of the linked data best practices in different topical domains. In *The Semantic Web–ISWC 2014*, pages 245–260. Springer, 2014.

[54] Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of sparql query optimization. In *Proceedings of the 13th International Conference on Database Theory*, pages 4–33. ACM, 2010.

[55] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *The Semantic Web–ISWC 2011*. Springer, 2011.

[56] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The semantic web revisited. *IEEE intelligent systems*, 21(3):96–101, 2006.

[57] Fuqi Song and Olivier Corby. Extended query pattern graph and heuristics-based sparql query planning. *Procedia Computer Science*, 60:302–311, 2015.

[58] Markus Stocker, Andy Seaborne, Abraham Bernstein, Christoph Kiefer, and Dave Reynolds. Sparql basic graph pattern optimization using selectivity estimation. In *Proceedings of the 17th international conference on World Wide Web*, pages 595–604. ACM, 2008.

[59] Petros Tsialiamanis, Lefteris Sidirourgos, Irini Fundulaki, Vassilis Christophides, and Peter Boncz. Heuristics-based query optimisation for sparql. In *Proceedings of the 15th International Conference on Extending Database Technology*, pages 324–335. ACM, 2012.

[60] Y. Tzitzikas, C. Alloca, C. Bekiari, Y. Marketakis, P. Fafalios, M. Doerr, N. Minadakis, T. Patkos, and L. Candela. Integrating Heterogeneous and Distributed Information about Marine Species through a Top Level Ontology. In *MTSR*, 2013.

[61] Jeffrey D Ullman. *Principles of database systems*. Galgotia publications, 1984.

[62] María-Esther Vidal, Edna Ruckhaus, Tomas Lampo, Amadís Martínez, Javier Sierra, and Axel Polleres. Efficiently joining group patterns in sparql queries. In *Extended Semantic Web Conference*, pages 228–242. Springer, 2010.

[63] Maria Papadaki Yannis Tzitzikas and Panagiotis Papadakos. An interactive 3d visualization for the lod cloud. `http://www.ics.forth.gr/isl/3DLod/`.